PXN20 Microcontroller Reference Manual

Devices Supported:

PXN2020
PXN2120

PXN20RM
Rev. 1
06/2011

How to Reach Us:

Home Page:
www.freescale.com

Web Support:
http://www.freescale.com/support

USA/Europe or Locations Not Listed:
Freescale Semiconductor, Inc.
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
1-800-521-6274 or +1-480-768-2130
www.freescale.com/support

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
www.freescale.com/support

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor China Ltd.
Exchange Building 23F
No. 118 Jianguo Road
Chaoyang District
Beijing 100022
China
+86 10 5879 8000
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or +1-303-675-2140
Fax: +1-303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

PXN20RM
Rev. 1
06/2011

Table of Contents

Chapter 1
Introduction

Chapter 2
Memory Map

Chapter 3
Signal Description

Chapter 4
Resets

Chapter 5
System Clock Description

Chapter 6
Clocks, Reset, and Power (CRP)

Chapter 7
Frequency Modulated Phase-Locked Loop (FMPLL)

Chapter 8
System Integration Unit (SIU)

Chapter 9
Boot Assist Module (BAM)

Chapter 10
Interrupts and Interrupt Controller (INTC)

Chapter 11
General-Purpose Static RAM (SRAM)

Chapter 12
Flash Memory Array and Control

Chapter 13
e200z6 Core (Z6)

Chapter 14
e200z0 Core (Z0)

Chapter 15
Semaphores

Chapter 16
AMBA Crossbar Switch (AXBS)

Chapter 17
Peripheral Bridge (AIPS-lite)

Chapter 18
Memory Protection Unit (MPU)

Chapter 19
Error Correction Status Module (ECSM)

Chapter 20
Software Watchdog Timer (SWT)

Chapter 21
System Timer Module (STM)

Chapter 22
Periodic Interrupt Timer (PIT)

Chapter 23
DMA Channel Multiplexer (DMA_MUX)

Chapter 24
Enhanced Direct Memory Access Controller (eDMA)

Chapter 25
Fast Ethernet Controller (FEC)

Chapter 26
FlexRay Communication Controller (FlexRAY)

Chapter 27
Media Local Bus (MLB)

Chapter 28
Enhanced Modular Input/Output Subsystem (eMIOS200)

Chapter 29
Controller Area Network (FlexCAN)

Chapter 30
Deserial – Serial Peripheral Interface (DSPI)

Chapter 31
Enhanced Serial Communication Interface (eSCI)

Chapter 32
Inter-Integrated Circuit Bus Controller Module (I^2C)

Chapter 33
Cross Triggering Unit (CTU)

Chapter 34
Analog-to-Digital Converter (ADC)

Chapter 35
IEEE 1149.1 Test Access Port Controller (JTAGC)

Chapter 36
Nexus Development Interface (NDI)

Appendix A
Memory Map

About This Book

This reference manual describes the PXN20 microcontroller family for software and hardware developers. Information regarding bus timing, signal behavior, and AC, DC, and thermal characteristics are detailed in the device data sheet (*PXN20 Microcontroller Data Sheet*).

The information in this book is subject to change without notice, as described in the disclaimers on the title page. As with any technical documentation, the reader needs to make sure to use the most recent version of the documentation.

To locate any published errata or updates for this document, refer to the world-wide web at http://www.freescale.com/powerpc.

Audience

This manual is intended for system software and hardware developers and applications programmers who want to develop products with the PXN20 microcontroller family. It is assumed that the reader understands operating systems, microprocessor system design, basic principles of software and hardware, and basic details of the Power Architecture™.

Organization

This document includes chapters that describe:

- The microcontroller as a whole
- The functionality of the individual modules on the microcontroller

When the microcontroller is specified as “PXN20,” the reader is instructed to apply this information to all of the microcontrollers specified on the front cover of this manual, unless individual device-specific details are provided in that chapter.

The following summary provides a brief description of the major sections of this manual:

- Chapter 1, Introduction, includes general descriptions of the modules and features incorporated in the device while focusing on new features.
- Chapter 2, Memory Map, provides a high-level listing of the PXN20 memory map.
- Chapter 3, Signal Description, summarizes the external signal functions, their static electrical characteristics, and pad configuration settings for the PXN20.
- Chapter 4, Resets, describes the reset sources available on the PXN20, including details on status flags and default configurations.
- Chapter 5, System Clock Description, describes the various clock sources that are available on the PXN20.

- Chapter 6, Clocks, Reset, and Power (CRP), describes the CRP block, which manages entry into, operation during, and exit from power-saving modes; and maintains all of the control logic that requires power when other portions of the PXN20 are powered down in power-saving modes.
- Chapter 7, Frequency Modulated Phase-Locked Loop (FMPLL), describes the features and function of the FMPLL module.
- Chapter 8, System Integration Unit (SIU), describes the SIU module, which controls MCU reset configuration, pad configuration, external interrupt, general-purpose I/O (GPIO), internal peripheral multiplexing, and the system reset operation.
- Chapter 9, Boot Assist Module (BAM), describes the BAM, which contains the MCU boot program code supporting the different booting modes for this device.
- Chapter 10, Interrupts and Interrupt Controller (INTC), summarizes the software and hardware interrupts for the PXN20 device.
- Chapter 11, General-Purpose Static RAM (SRAM), describes the on-chip static RAM (SRAM) implementation, covers general operations, configuration, and initialization. It also provides information and examples of how to minimize power consumption when using the SRAM.
- Chapter 12, Flash Memory Array and Control, describes the flash memory block and the flash memory controller.
- Chapter 13, e200z6 Core (Z6), describes the organization of the e200z6 Power processor core and gives an overview of the programming models as they are implemented on the device. The e200z6 is the main processor core on the PXN20.
- Chapter 14, e200z0 Core (Z0), describes the organization of the e200z0 Power processor core and an overview of the programming models as they are implemented on the device. The e200z0 serves as an input/output (I/O) processor on the PXN20.
- Chapter 15, Semaphores, describes the module that lets each processor know which processor has control of common memory.
- Chapter 16, AMBA Crossbar Switch (AXBS), describes the multi-port crossbar switch that supports simultaneous connections between six master ports and six slave ports.
- Chapter 17, Peripheral Bridge (AIPS-lite), describes the interface between the system bus and lower bandwidth peripherals.
- Chapter 18, Memory Protection Unit (MPU), describes the block that provides hardware access control for all memory references generated in the PXN20.
- Chapter 19, Error Correction Status Module (ECSM), describes the ECSM block, which provides monitoring and control functions to report memory errors and apply error-correcting code (ECC) implementations.
- Chapter 20, Software Watchdog Timer (SWT), describes a hardware-based timer that can be implemented to prevent software runaway.
- Chapter 21, System Timer Module (STM), describes the timer control module.
- Chapter 22, Periodic Interrupt Timer (PIT), describes an array of timers that can be used to initiate interrupts and trigger DMA channels.
- Chapter 23, DMA Channel Multiplexer (DMA_MUX), describes the DMA multiplexer block implemented on the PXN20.

- Chapter 24, Enhanced Direct Memory Access Controller (eDMA), describes the enhanced DMA controller implemented on the PXN20.
- Chapter 25, Fast Ethernet Controller (FEC), describes the feature set, operation, and programming model of the FEC block.
- Chapter 26, FlexRay Communication Controller (FlexRAY), describes the FlexRay communication controller on the PXN20 that implements the *FlexRay Communications System Protocol Specification, Version 2.1 Rev A.*
- Chapter 27, Media Local Bus (MLB), describes the MLB module, a multiplexed bus controller that transfers multimedia data between the MOST ring and supporting system level ICs.
- Chapter 28, Enhanced Modular Input/Output Subsystem (eMIOS200), describes the eMIOS module, which provides timed I/O channels for communications with off-chip devices.
- Chapter 29, Controller Area Network (FlexCAN), describes the FlexCAN module, a communication controller implementing the CAN protocol according to Bosch Specification version 2.0B and ISO Standard 11898.
- Chapter 30, Deserial – Serial Peripheral Interface (DSPI), describes the DSPI block, which provides a synchronous serial interface for communication between the PXN20 and external devices.
- Chapter 31, Enhanced Serial Communication Interface (eSCI), describes the eSCI interface, which allows asynchronous serial communications with off-chip peripheral devices.
- Chapter 32, Inter-Integrated Circuit Bus Controller Module (I2C), describes the I^2C module, including I^2C protocol, clock synchronization, and I^2C programming model registers.
- Chapter 33, Cross Triggering Unit (CTU), describes the CTU block, which converts the events generated by the eMIOS into ADC conversion requests. It also has a PIT channel. The CTU interfaces between the eMIOS/PIT and the ADC and converts the events generated by the eMIOS into ADC conversion requests.
- Chapter 34, Analog-to-Digital Converter (ADC), describes the ADC module implemented on the PXN20.
- Chapter 35, IEEE 1149.1 Test Access Port Controller (JTAGC), describes configuration and operation of the Joint Test Action Group (JTAG) controller implementation. It describes those items required by the IEEE® 1149.1 standard and provides additional information specific to the device. For internal details and sample applications, see the IEEE 1149.1 document.
- Chapter 36, Nexus Development Interface (NDI), describes the Nexus Development Interface (NDI) block, which provides real-time development support capabilities for the PXN20 in compliance with the IEEE-ISTO 5001-2003 standard.
- Appendix A, Memory Map, provides a detailed listing of the memory-mapped registers for the PXN20.

Suggested Reading

This section lists additional reading that provides background for the information in this manual as well as general information about Power Architecture

General Information

Useful information about the Power Architecture and computer architecture in general:

- *Programming Environments Manual for 32-Bit Implementations of the PowerPC™ Architecture* (MPCFPE32B)
- *Using Microprocessors and Microcomputers: The Motorola Family*, William C. Wray, Ross Bannatyne, Joseph D. Greenfield
- *Computer Architecture: A Quantitative Approach*, Second Edition, by John L. Hennessy and David A. Patterson.
- *Computer Organization and Design: The Hardware/Software Interface*, Second Edition, David A. Patterson and John L. Hennessy.

Power Architecture Documentation

Power Architecture documentation is available from the sources listed on the back cover of this manual, as well as our web site, http://www.freescale.com/powerpc.

- Reference manuals (formerly called user's manuals)—These books provide details about individual Power Architecture implementations and are intended to be used in conjunction with the *PowerPC Programmers Reference Manual*.
- Addenda/errata to reference manuals—Because some processors have follow-on parts, an addendum is provided that describes the additional features and functionality changes. Also, if mistakes are found within a reference manual, an errata document will be issued before the next published release of the reference manual. These addenda/errata are intended for use with the corresponding reference manuals.
- Data sheets—Data sheets provide specific information regarding pin-out diagrams, bus timing, signal behavior, and AC, DC, and thermal characteristics, as well as other design considerations.
- Product briefs—Each device has a product brief that provides an overview of its features. This document is roughly equivalent to the overview (Chapter 1) of a device's reference manual.
- Application notes—These short documents address specific design issues useful to programmers and engineers working with Freescale Semiconductor processors.

Additional literature is published as new processors become available. For a current list of Power Architecture documentation, refer to http://www.freescale.com/powerpc.

Conventions

This document uses the following notational conventions:

cleared/set	When a bit takes the value 0, it is said to be cleared; when it takes a value of 1, it is said to be set.

reserved	When a bit or address is reserved, it should not be written. If read, its value is not guaranteed. Reading or writing to reserved bits or addresses may cause unexpected results.
MNEMONICS	In text, instruction mnemonics are shown in uppercase.
`mnemonics`	In code and tables, instruction mnemonics are shown in lowercase.
italics	Italics indicate variable command parameters. Book titles in text are set in italics.
0x	Prefix to denote hexadecimal number
0b	Prefix to denote binary number
REG[FIELD]	Abbreviations for registers are shown in uppercase. Specific bits, fields, or ranges appear in brackets. For example, RAMBAR[BA] identifies the base address field in the RAM base address register.
nibble	A 4-bit data unit
byte	An 8-bit data unit
halfword	A 16-bit data unit[1]
word	A 32-bit data unit
doubleword	A 64-bit data unit
x	In some contexts, such as signal encodings, x (without italics) indicates a “don’t care” condition.
x	With italics, used to express an undefined alphanumeric value (e.g., a variable in an equation); or a variable alphabetic character in a bit, register, or module name (e.g., DSPI_*x* could refer to DSPI_A or DSPI_B).
n	Used to express an undefined numerical value; or a variable numeric character in a bit, register, or module name (e.g., EIF*n* could refer to EIF1 or EIF0).
~	NOT logical operator
&	AND logical operator
|	OR logical operator
||	Field concatenation operator
$\overline{\text{OVERBAR}}$	An overbar indicates that a signal is active-low.

Register Figure Conventions

This document uses the following conventions for the register reset values in register figures:

—	Bit value is undefined at reset.
U	Bit value is unchanged by reset. Previous value preserved during reset.
[*signal_name*]	Reset value is determined by the polarity of the indicated signal.

1. The only exceptions to this appear in the discussion of serial communication modules that support variable-length data transmission units. To simplify the discussion these units are referred to as words regardless of length.

The following descriptions are used in register bit field description tables:

R W	0	Indicates a reserved bit field in a memory-mapped register. These bits are always read as 0.
R W	1	Indicates a reserved bit field in a memory-mapped register. These bits are always read as 1.
R W	FIELDNAME	Indicates a read/write bit in a memory-mapped register.
R W	FIELDNAME	Indicates a read-only bit field in a memory-mapped register.
R W	 FIELDNAME	Indicates a write-only bit field in a memory-mapped register.
R W	FIELDNAME w1c	Write 1 to clear: indicates that writing a 1 to this bit field clears it.
R W	FIELDNAME r1c	Read to clear: indicates that reading this bit field clears it, regardless of its returned value.
R W	0 FIELDNAME	Indicates a self-clearing bit.

Acronyms and Abbreviations

Table i lists acronyms and abbreviations used in this document.

Table i. Acronyms and Abbreviated Terms

Term	Meaning
ADC	Analog-to-digital converter
ALU	Arithmetic logic unit
BIST	Built-in self test
DMA	Direct memory access
EA	Effective address
FIFO	First-in, first-out
GPIO	General-purpose I/O
I^2C	Inter-integrated circuit
IEEE	Institute for Electrical and Electronics Engineers

Table i. Acronyms and Abbreviated Terms (continued)

Term	Meaning
JEDEC	Joint Electron Device Engineering Council
JTAG	Joint Test Action Group
LIFO	Last-in, first-out
LSB	Least-significant byte
LVI	Low-voltage interrupt
lsb	Least-significant bit
MAC	Multiply accumulate unit, also Media access controller
MSB	Most-significant byte
msb	Most-significant bit
Mux	Multiplex
NOP	No operation
OEP	Operand execution pipeline
PC	Program counter
PLIC	Physical layer interface controller
PLL	Phase-locked loop
POR	Power-on reset
RISC	Reduced instruction set computing
Rx	Receive
SOF	Start of frame
TAP	Test access port
TTL	Transistor transistor logic
Tx	Transmit
UART	Universal asynchronous/synchronous receiver transmitter
USB	Universal serial bus

Terminology Conventions

Table ii shows terminology conventions used throughout this document.

Table ii. Notational Conventions

Instruction	Operand Syntax
Opcode Wildcard	
cc	Logical condition (example: NE for not equal)

Table ii. Notational Conventions (continued)

Instruction	Operand Syntax
Register Specifications	
An	Any address register n (example: A3 is address register 3)
Ay,Ax	Source and destination address registers, respectively
Dn	Any data register n (example: D5 is data register 5)
Dy,Dx	Source and destination data registers, respectively
Rc	Any control register (example VBR is the vector base register)
Rm	MAC registers (ACC, MAC, MASK)
Rn	Any address or data register
Rw	Destination register w (used for MAC instructions only)
Ry,Rx	Any source and destination registers, respectively
Xi	Index register i (can be an address or data register: Ai, Di)
Miscellaneous Operands	
#<data>	Immediate data following the 16-bit operation word of the instruction
<ea>	Effective address
<ea>y,<ea>x	Source and destination effective addresses, respectively
<label>	Assembly language program label
<list>	List of registers for MOVEM instruction (example: D3–D0)
<shift>	Shift operation: shift left (<<), shift right (>>)
<size>	Operand data size: byte (B), word (W), longword (L)
bc	Instruction and data caches
dc	Data cache
ic	Instruction cache
# <vector>	Identifies the 4-bit vector number for trap instructions
<>	identifies an indirect data address referencing memory
<xxx>	identifies an absolute address referencing memory
d*n*	Signal displacement value, *n* bits wide (example: d16 is a 16-bit displacement)
SF	Scale factor (x1, x2, x4 for indexed addressing mode, <<1n>> for MAC operations)
Operations	
+	Arithmetic addition or postincrement indicator
–	Arithmetic subtraction or predecrement indicator
x	Arithmetic multiplication

Table ii. Notational Conventions (continued)

Instruction	Operand Syntax
/	Arithmetic division
~	Invert; operand is logically complemented
&	Logical AND
|	Logical OR
^	Logical exclusive OR
<<	Shift left (example: D0 << 3 is shift D0 left 3 bits)
>>	Shift right (example: D0 >> 3 is shift D0 right 3 bits)
→	Source operand is moved to destination operand
←→	Two operands are exchanged
sign-extended	All bits of the upper portion are made equal to the high-order bit of the lower portion
If <condition> then <operations> else <operations>	Test the condition. If true, the operations after then are performed. If the condition is false and the optional else clause is present, the operations after else are performed. If the condition is false and else is omitted, the instruction performs no operation. Refer to the Bcc instruction description as an example.
Subfields and Qualifiers	
{}	Optional operation
()	Identifies an indirect address
d_n	Displacement value, n-bits wide (example: d_{16} is a 16-bit displacement)
Address	Calculated effective address (pointer)
Bit	Bit selection (example: Bit 3 of D0)
lsb	Least significant bit (example: lsb of D0)
LSB	Least significant byte
LSW	Least significant word
msb	Most significant bit
MSB	Most significant byte
MSW	Most significant word

Chapter 1
Introduction

1.1 Overview

The PXN20 products are compatible 32-bit microcontrollers built on Power Architecture® technology. This document describes the available features, and highlights important characteristics of the devices.

The PXN20 products are designed to address the need for single chip industrial networking applications and are tailored to address the need for high performance and high memory size while keeping the power consumption low. Their core and bus architecture offer high performance processing optimized for managing intensive data exchanges between different types of communication protocols. It capitalizes on the available development infrastructure of current Power Architecture devices and will be supported with software drivers and an operating system to assist with user implementations.

The PXN20 devices have two levels of memory hierarchy, a 32 KB unified cache, and 2 MB of internal flash. The PXN20 has 128 KB on-chip L2 SRAM and the PXN21 has 592 KB on-chip L2 SRAM. Refer to Table 1 for specific memory and feature sets of the family members.

1.2 PXN20 Features

Table 1-1 provides a summary of the different members of the PXN20 family and their features. This information is intended to provide an understanding of the range of functionality offered by this family of devices.

Table 1-1. PXN20 Family Feature Set

Feature	PXN20	PXN21
Central Processing Unit (CPU)	e200z650	e200z650
Cache	32K, 4/8way	32K, 4/8way
Floating Point Unit (FPU)	Yes	Yes
Signal Processing Engine (SPE)	Yes	Yes
Memory Management Unit (MMU)	32 entry	32 entry
CPU Execution Speed	Static, 116 MHz	Static, 116 MHz
Input/Output Processor (IOP)	e200z0	e200z0
IOP Execution Speed	1/2 CPU execution speed	1/2 CPU execution speed
Flash with ECC	2 MB	2 MB
Data Flash Block	8x16 KB	8x16 KB
RAM with ECC	592 KB	128 KB

Table 1-1. PXN20 Family Feature Set

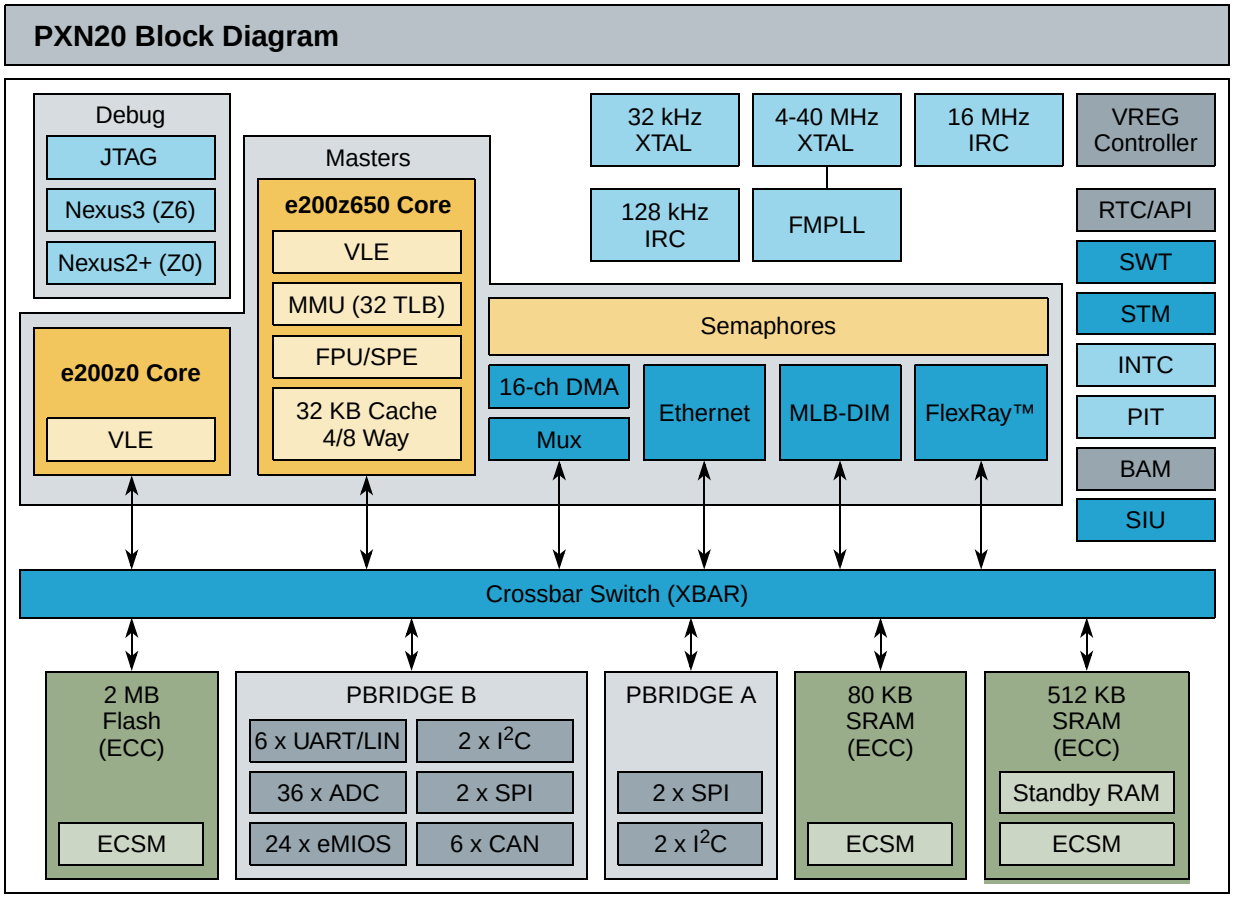
Feature	PXN20	PXN21
Memory Protection Unit (MPU)	No	16 entry
Direct Memory Access Unit (eDMA)	16 Channel	32 Channel
Ethernet (FEC)	Yes	No
MediaLB (MLB-DIM)	Yes	No
FlexRay Controller	Yes (128 Message Buffers)	No
Analog-to-Digital Converter (ADC)	36 internal channels, 10-bit Supports 32 external channels	64 internal channels, 10-bit Supports 32 external channels
Total Timer I/O (eMIOS200)	24 channels, 16-bit	32 channels, 16-bit
Cross Trigger Unit (CTU)	No	Yes
Asynchronous Serial Interfaces (UART)	6	12
Synchronous Serial Interfaces (SPI)	4	4
Controller Area Network (CAN) Controller	6	5
Inter-Integrated Circuit (I^2C) Controller	4	4
Frequency Modulated PLL (FMPLL)	Yes	Yes
4 – 40 MHz XTAL Oscillator	Yes	Yes
16 MHz IRC Oscillator	Yes	Yes
32 kHz XTAL Oscillator	Yes	Yes
128 kHz IRC Oscillator	Yes	Yes
Real Time Counter/ Autonomous Periodic Interrupts (RTC/API)	Yes	Yes
Periodic Interrupt Timer (PIT)	8	8
System Timer Module (STM)	Yes	Yes
Software Watchdog Timer (SWT)	Yes	Yes
General-Purpose I/O (GPIO)	155	155
Clock Monitor (FMPLL)	Yes	Yes
JTAG	Yes	Yes
Nexus Debug (Only supported on emulation package)	Nexus3 (e200Z6) Nexus2+ (e200Z0)	Nexus3 (e200Z6) Nexus2+ (e200Z0)
Production Package	208 MAPBGA	208 MAPBGA
Emulation Package (for development use only)	256 MAPBGA	256 MAPBGA

Throughout this book, the e200z650n3e core may also be referred to as the Z6 or the e200z6. In the context of the PXN20 device, these terms are interchangeable. Refer to the *e200z6 PowerPC™ Core Reference Manual* for more information on the e200z6 core.

Similarly, the e200z0 core is also referred to as the Z0.

1.3 PXN20 Block Diagram

Figure 1-1 shows a top-level block diagram of the PXN20.

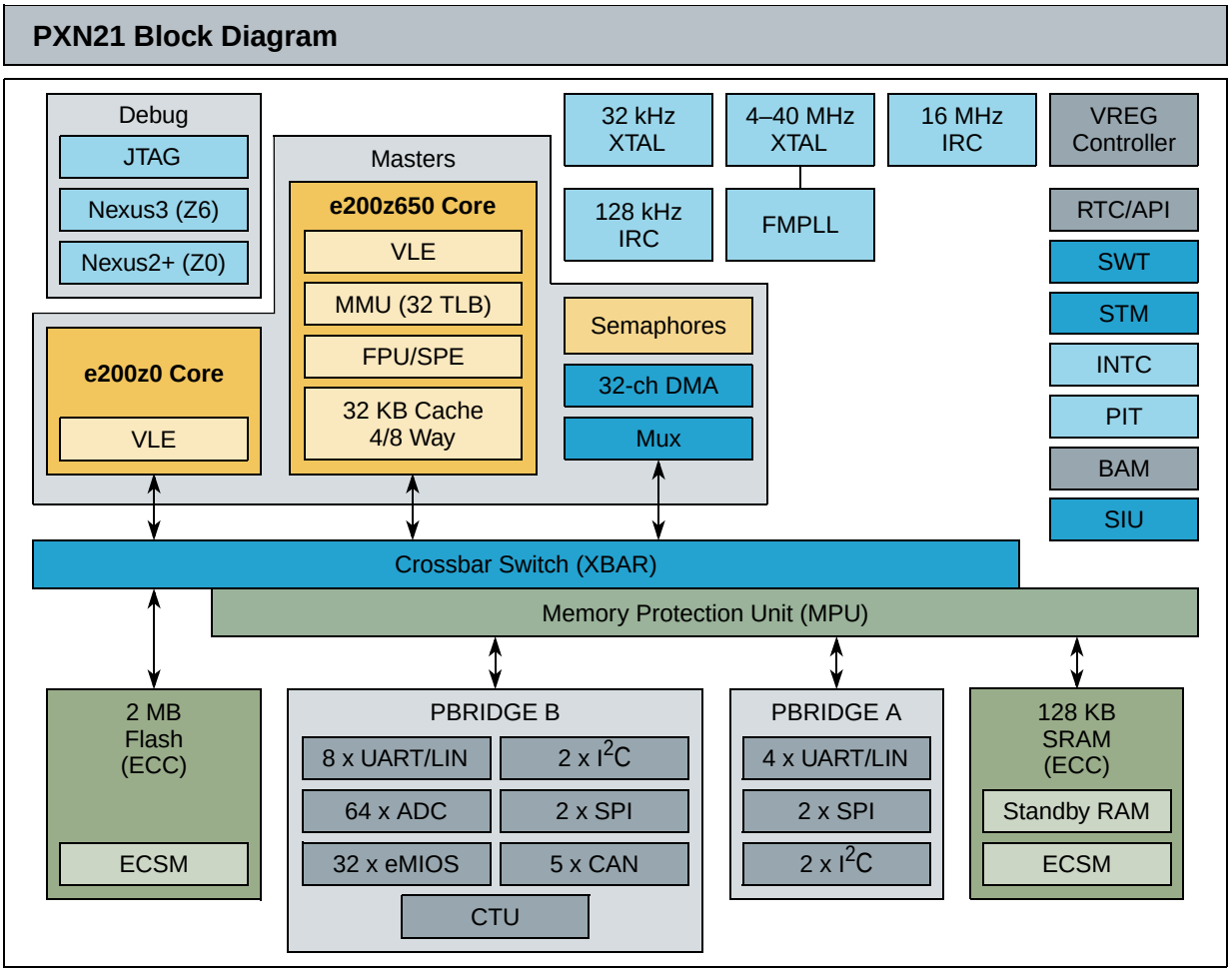


ADC – Analog-to-digital converter
BAM – Boot assist module
CAN – Controller area network controller
ECC – Error correction code
ECSM – Error correction status module
eMIOS – Timed input/output
eDMA – Enhanced direct memory access controller
FEC – Fast ethernet controller
FlexRay™ – FlexRay bus controller
FMPLL – Frequency-modulated phase-locked loop
I²C – Inter-integrated circuit controller
INTC – Interrupt controller
JTAG – Joint Test Action Group interface
MLB-DIM – Media Local Bus device interface module
NDI – Nexus debug interface
PBRIDGE – Peripheral I/O bridge
PIT – Periodic interrupt timer
RTC – Real time clock
SIU – System integration unit
SPI – Serial peripheral interface controller
STM – System timer module
SWT – Software watchdog timer
UART/LIN – Universal asynchronous receiver/transmitter/ local interconnect network
VREG – Voltage regator

Figure 1-1. PXN20 Block Diagram

1.4 PXN21 Block Diagram

Figure 1-2 shows a top-level block diagram for the PXN21.



ADC – Analog-to-digital converter
BAM – Boot assist module
CAN – Controller area network controller
CTU – Cross triggering unit
ECC – Error correction code
ECSM – Error correction status module
eDMA – Enhanced direct memory access controller
eMIOS – Timed input/output
FMPLL – Frequency-modulated phase-locked loop
I^2C – Inter-integrated circuit controller
INTC – Interrupt controller
JTAG – Joint Test Action Group interface
MPU – Memory protection unit
NDI – Nexus debug interface
PBRIDGE – Peripheral I/O bridge
PIT – Periodic interrupt timer
RTC – Real time clock
SIU – System integration unit
SPI – Serial peripheral interface controller
STM – System timer module
SWT – Software watchdog timer
UART/LIN – Universal asynchronous receiver/transmitter/ local interconnect network
VREG – Voltage regulator

Figure 1-2. PXN21 Block Diagram

1.5 Critical Performance Parameters

The critical performance parameters of the PXN20 devices feature the following:

- Fully static design operation up to a maximum of 116 MHz, based on 105 °C ambient
- Temperature range –40° to 105 °C ambient temperature
- Low power design
  - — Designed for dynamic power management of core and peripherals
  - — Software-controlled clock gating of peripherals
  - — Simple power domains to minimize leakage in low power modes
  - — Internal voltage regulator (VREG) enables operation with a single input voltage
  - — 3.3 V / 5 V supply (nominal)
  - — External Ballast control
- ADC analog supply range 3.0 V – 5.5 V
- Low voltage detect circuit implemented
- Configurable pins
  - — Selectable pull-up, pull-down, or no pull on all GPIO pins
  - — Selectable open-drain pin

1.5.1 Low Power Operation

The PXN20 devices have one dynamic power mode and one static power mode:

- Low-power modes use clock gating to halt the clock for all or part of the device
- The lowest power mode also uses power gating to automatically turn off the power supply to parts of the device to minimize leakage
- Dynamic power mode: RUN
  - — RUN mode is the main operating mode where the entire device is powered and clocked and where most processing activity is done
- Static power mode: SLEEP
  - — SLEEP mode halts the clock to the entire device and turns off the power to the majority of the chip in order to offer the lowest power consumption mode of the PXN20. In SLEEP mode the contents of the cores, on-chip peripheral registers and part of the volatile memory are not held. The device can be awakened from up to 32 I/O pins, a reset, or from an internal periodic wake-up. It is also possible to enable the 16 MHz IRC, the 4–40 MHz XTAL, 128 kHz IRC and the 32 kHz XTAL.
    - – SLEEP1 mode retains 32 KB of the on-chip RAM
    - – SLEEP2 mode retains the 64 KB of the on-chip RAM
    - – SLEEP3 mode retains 128 KB of the on-chip RAM
  - — Fast wake-up using the on-chip 16 MHz IRC allows rapid execution from RAM on exit from low power modes
    - – In SLEEP mode, a 4 – 40 MHz XTAL can be enabled to continue to run

  - In SLEEP mode, the 16 MHz IRC can be enabled to continue to run and may be selected to clock the RTC and API
  - In SLEEP mode, the 128 kHz IRC can be enabled to run and may be selected to clock the RTC and API
  - In Sleep mode the 32 kHz XTAL can be enabled to run and may be selected to clock the RTC and API
- Up to 32 external pins for wake-up from low power modes
- Input filters available on all wake-up pins to minimize false wakeups due to noise
- Rapid exit from low power mode with fast startup internal voltage regulator

1.6 Packages

PXN20 family members are offered in the following package types:

- 208-ball MAPBGA, 1mm ball pitch, 17mm × 17mm outline for production
- 256-ball MAPBGA 1mm ball pitch 17mm × 17mm outline for emulation, providing access to full Nexus port without sacrificing GPIO functionality (not available for production)

1.7 Module Features

The following sections provide details of the modules implemented on the PXN20 family.

1.7.1 High Performance e200z650 Core Processor (CPU)

32-bit CPU built on Power Architecture® technology

- Freescale Variable Length Encoding (VLE) enhancements for code size footprint reduction
- Thirty-two 64-bit general-purpose registers (GPRs)
- Memory management unit (MMU) with 32-entry fully-associative translation look-aside buffer (TLB)
- Branch processing unit
- Fully pipelined load/store unit
- 32 KB unified cache with line locking
  - 4/8-way set associative
  - Two 32-bit fetches per clock
  - Eight-entry store buffer
  - Way locking
  - Supports assigning cache as instruction or data only on a per way basis
  - Supports tag and data parity
- Vectored interrupt support
- Very low interrupt latency
- Reservation instructions for implementing read-modify-write constructs (internal SRAM and flash)

- Signal processing engine (SPE) auxiliary processing unit (APU) operating on 64-bit general purpose registers
- Floating point
  - IEEE® 754 compatible with software wrapper
  - Single precision in hardware; double precision with software library
  - Conversion instructions between single precision floating point and fixed point
- Wait instruction
- Extensive system development support through Nexus debug module

1.7.2 I/O Processor High Performance e200z0 Core (IOP)

The IOP supports the following features:

- High performance, low cost e200z0 core processor for managing peripherals and interrupts
- Single issue 4-stage pipelined in-order execution, 32-bit Power Architecture® CPU
- Variable length encoding (VLE), allowing mixed 16-bit and 32-bit instructions
  - Results in efficient code size footprint
  - Minimizes impact on performance
- Branch processing acceleration using lookahead instruction buffer
- Load/store unit
  - 1-cycle load latency
  - Misaligned access support
  - No load-to-use pipeline bubbles
- Thirty-two 32-bit general purpose registers (GPRs)
- Hardware vectored interrupt support
- Reservation instructions for implementing read-modify-write constructs
- Multi-cycle divide (divw) and load multiple (lmw) store multiple (smw) multiple class instructions, can be interrupted to prevent increases in interrupt latency
- Extensive system development support through Nexus debug port

1.7.3 On-Chip Flash

On-chip flash on the PXN20 devices features the following:

- 2 MB burst flash memory
  - Flash partitioning: 4 × 16 KB; 4 × 16 KB; 2 × 64 KB; 2 × 128 KB; 6 × 256 KB
  - 16 KB shadow flash blocks
  - Typical flash access time: 0 wait-state for buffer hits, 3 wait-states for page buffer miss at 116 MHz
  - 64-bit ECC with single-bit correction, double-bit detection for data integrity
- Dual flash ports to minimize access contention between main core and IOP

  - Each port supported with separate page buffers
- Flash page buffers to improve access time to code and data held in flash
  - 4 × 128-bit page buffers with programmable prefetch control for flash access
  - Page buffers can be allocated for code-only, fixed partitions of code and data, all available for any access
- Censorship protection scheme to prevent flash content visibility
- EE emulation supported by small 16 KB flash blocks in main array with multiple read while write partitions
- Hardware managed flash writes, erase and verify sequence
- Supports flash writes using internal 16 MHz RC oscillator
- Flash partitioning:

Table 2. Flash Partitioning

	PXN20
	2 MB
Flash_Base + 0x0000_0000	16 KB
Flash_Base + 0x0000_4000	16 KB
Flash_Base + 0x0000_8000	16 KB
Flash_Base + 0x0000_C000	16 KB
Flash_Base + 0x0001_0000	16 KB
Flash_Base + 0x0001_4000	16 KB
Flash_Base + 0x0001_8000	16 KB
Flash_Base + 0x0001_C000	16 KB
Flash_Base + 0x0002_0000	64 KB
Flash_Base + 0x0003_0000	64 KB
Flash_Base + 0x0004_0000	128 KB
Flash_Base + 0x0006_0000	128 KB
Flash_Base + 0x0008_0000	256 KB
Flash_Base + 0x000C_0000	256 KB
Flash_Base + 0x0010_0000	256 KB
Flash_Base + 0x0014_0000	256 KB
Flash_Base + 0x0018_0000	256 KB
Flash_Base + 0x001C_0000	256 KB
Shadow Block	16 KB

- Error correction status
  - Configurable error-correcting codes (ECC) reporting for RAM and flash

1.7.4 On-Chip SRAM

On-chip SRAM on the PXN20 family features the following:

- Up to 592/128 KB general purpose RAM
- Two RAM blocks implemented on separate crossbar ports to reduce arbitration events for high access master to on-chip RAM.
  - — One port with 80 KB (PXN20 only)
  - — One port with 512/128 KB RAM
- Typical SRAM access time: 0 wait-state for reads and 32-bit writes; 1 wait-state for 8- and 16-bit writes if back to back with a read to same memory block
- 32-bit ECC with single-bit correction, double bit detection for data integrity
- Supports byte (8-bit), half word (16-bit), and word (32-bit) writes for optimal use of memory
- User transparent ECC encoding and decoding for byte, half word, and word accesses

1.7.5 On-Chip Voltage Regulator (VREG)

The on-chip voltage regulator includes the following features:

- Single supply device
- 3.3 V / 5 V (nominal) input supply voltage supported
- Supports I/O levels independent of main supply
  - — MLB has separate supply pins to support down to 2.5 V (nominal) operation
  - — Multiple I/O domains with separate supply pins
- Low voltage detectors (LVD) supported on internal supplies
- Cold crank operation supported without triggering LVDs

1.7.6 Fast Ethernet Controller (FEC)

The FEC incorporates the following features

- Support for 3 different physical interfaces
  - — 100 Mbps IEEE 802.3 MII
  - — 10 Mbps IEEE 802.3 MII
  - — 10 Mbps 7-wire interface (industry standard)
- Built in FIFO and DMA controller
- IEEE 802.3 MAC (compliant with IEEE 802.3 1998 edition)
- Programmable max frame length supports IEEE 802.1 VLAN tags and priority
- IEEE 802.3 full duplex flow control
- Support for full duplex operation (200 Mbps throughput) with a system clock of 100 MHz using the external TX_CLK or RX_CLK
- Support for full duplex operation(100 Mbps throughput) with a system clock of 50 MHz using the external TX_CLK or RX_CLK

- Retransmission from transmit FIFO following a collision (no system bus utilization)
- Automatic internal flushing of the receive FIFO for runts (collision fragments) and address recognition rejects (no system bus utilization)
- Address recognition
- RMON and IEEE statistics
- Interrupts for network activity and error conditions

NOTE

The FEC is available on the PXN21 only.

1.7.7 Analog to Digital Converter Module (ADC)

The PXN20 ADC features the following:

- 10-bit A/D resolution
- 0–V_{DD} common mode conversion range
- Supports conversions speeds of up to 1 µs
- Internally multiplexed channels
  - 10-bit ± 2 least significant bits (LSB) accuracy (TUE) available for 16 channels
  - 10-bit ± 3 LSB accuracy (TUE) available for remaining channels
  - Dedicated result register available for every internally muxed channel
- Externally multiplexed channels
  - Internal control to support generation of external analog multiplexor selection
  - Four internal channels optionally used to support externally multiplex inputs, providing transparent control for additional ADC channels
  - Each of the four channels supports up to 8 externally muxed inputs
- Three independently configurable sample and conversion times for high occurrence channels, internally muxed channels and externally muxed channels
- Right-aligned result format
- Support for one-shot, scan and injection conversion modes
- Traceability of each channels with conversion result.
- Injection mode status bit implemented on adjacent 16-bit register for each result
- Independently configurable parameters for channels:
  - Offset refresh
  - Sampling
- Cross Triggering support (PXN21 only)
  - Internal conversion triggering from periodic interrupt timer (PIT) or timed I/O module (eMIOS200) via Cross Triggering Unit (CTU)
  - One input pin configurable as external conversion trigger source
- Four configurable analog comparator channels offering range comparison with triggered alarm

- Supports operation of ADC using internal 16 MHz RC oscillator
- All unused analog pins available as general purpose input pins
- Selected unused analog pins available as general purpose output pins
- Power-down mode
- Supports for DMA transfer of results

1.7.8 Cross Triggering Unit (CTU)

The CTU features the following:

- Collection of 9 bit timers with an exponential prescaler able to generate the trigger event for ADC conversions
- 9-bit down counters counting from a programmable start value to 0
- Different counters associated with different channel groups
- Channel group is defined based on PWM channel clock
- Different delay value for each eMIOS flag/PIT event
- 4-bit programmable exponential prescaler
- Single cycle delayed trigger output. Trigger output is a combination of 64 input flags/events connected to different timers in the system
- Maskable interrupt generation whenever a trigger output is generated
- Event configuration registers dedicated to UC flag/triggering event
- Acknowledgement signal to eMIOS for clearing the flag
- Synchronization with ADC to avoid collision

NOTE

The CTU is available on the PXN21 only.

1.7.9 Serial Communication Interface Module (UART)

The PXN20 devices include up to two UART modules and support UART Master mode, UART Slave mode and UART mode. The modules are UART state machine compliant to the UART 1.3 and 2.0 and 2.1 Specifications and handle UART frame transmission and reception without CPU intervention.

The serial communication interface module offers the following:

- UART features:
  - Full-duplex operation
  - Standard non return-to-zero (NRZ) mark/space format
  - Data buffers with 4-byte receive, 4-byte transmit
  - Configurable word length (8-bit or 9-bit words)
  - Error detection and flagging
    - Parity, noise and framing errors
  - Interrupt driven operation with 4 interrupts sources

- Separate transmitter and receiver CPU interrupt sources
- 16-bit programmable baud-rate modulus counter and 16-bit fractional
- 2 receiver wake-up methods

- LIN features:
  - Autonomous LIN frame handling
  - Message buffer to store identifier and up to eight data bytes
  - Supports message length of up to 64 bytes
  - Detection and flagging of LIN errors
  - Sync field; Delimiter; ID parity; Bit, Framing; Checksum and Timeout errors
  - Classic or extended checksum calculation
  - Configurable Break duration of up to 36-bit times
  - Programmable Baud rate prescalers (13-bit mantissa, 4-bit fractional)
  - Diagnostic features
    - Loop back
    - Self Test
    - LIN bus stuck dominant detection
  - Interrupt driven operation with 16 interrupt sources
  - LIN slave mode features
    - Autonomous LIN header handling
    - Autonomous LIN response handling
    - Discarding of irrelevant LIN responses using up to 16 ID filters

1.7.10 Controller Area Network Module (CAN)

The enhanced CAN module features the following:

- Compliant with CAN protocol specification, Version 2.0B active
- 64 mailboxes, each configurable as transmit or receive
  - Mailboxes configurable while module remains syncronised to CAN bus
- Transmit features
  - Arbitration scheme according to message ID, message buffer number or local priority
  - Internal arbitration to guarantee no inner priority inversion
  - Multiple transmit buffers to avoid outer priority inversion
  - Transmit abort procedure and notification
- Receive features
  - Individual programmable filters for each mailbox
  - Hardware FIFO can be enabled
    - 8 mailboxes can be configured to provide a 6-entry receive FIFO and 8 programmable acceptance filters

- Programmable clock source
  - System clock
  - Direct oscillator clock to avoid PLL jitter
  - Listen only mode capabilities

1.7.11 Inter-IC Communications Module (I^2C)

The I^2C module features the following:

- Two-wire bi-directional serial bus for on-board communications
- Compatibility with I^2C bus standard
- Multimaster operation
- Software-programmable for one of 256 different serial clock frequencies
- Software-selectable acknowledge bit
- Interrupt-driven, byte-by-byte data transfer
- Arbitration-lost interrupt with automatic mode switching from master to slave
- Calling address identification interrupt
- Start and stop signal generation/detection
- Repeated START signal generation
- Acknowledge bit generation/detection
- Bus-busy detection

1.7.12 Serial Peripheral Interface Module (SPI)

The PXN20 SPI features the following:

- Full duplex, synchronous transfers
- Master or slave operation
- Programmable master bit rates
- Programmable clock polarity and phase
- End-of-transmission interrupt flag
- Programmable transfer baud rate
- Programmable data frames from 4 to 16 bits
- Up to 24 chip select lines available (6 per SPI module); the number available at any time is dependent on package and pin multiplexing.
- Up to 4 independently configurable transfer types can be configured for each SPI using the clock and transfer attributes registers
- Chip select strobe available as alternate function on one of the chip select pins for de-glitching
- FIFOs for buffering up to 4 transfers on the transmit and receive side
- General purpose I/O functionality on pins when not used for SPI
- Queueing operation possible through use of eDMA

- Serialization of selected sources (eMIOS channels and Phantom ports in SIU)

1.7.13 Enhanced Modular Input Output System (Timers - eMIOS200)

The PXN20 family implement a scaled-down version of the eMIOS module:

- Supports timed I/O channels with 16-bit counter resolution
- Buffered updates
- Support for shifted PWM outputs to minimize occurrence of concurrent edges
- Supports configurable trigger outputs for ADC conversions for synchronization to channel output waveforms
- Edge aligned output pulse width modulation
  - Programmable pulse period and duty cycle
  - Supports 0% and 100% duty cycle
  - Shared or independent time bases
- Up to 32 [1] single action channels offering input capture and output compare functions
- Up to 32 [1] dual action channels offering output pulse width modulation,
- Up to 13 [1] output pulse width and frequency modulation and center aligned output PWM channels with dead time.
- Up to 5 [1] modulus up/down counters that can be used to drive counter buses.
- DMA transfer support available

1.7.14 Periodic Interrupt Timer Module (PIT)

The PIT features the following:

- Up to 8 general purpose interrupt timers
- Up to 2 interrupt timers for triggering ADC conversions
- 32-bit counter resolution
- Clocked by system clock frequency

1.7.15 System Timer Module (STM)

One STM supporting

- One 32-bit up counter with 8-bit prescaler
- Four 32-bit compare channels running off the same up-counter
- Independent interrupt source for each channel
- Clocked by the main system clock
- Instantiated in the same CPU clock domain
- Counter can be stopped in debug mode

1. Depends on pin muxing and product derivative

1.7.16 Enhanced Direct Memory Access Controller (eDMA)

The following summarizes the PXN20 implementation of the eDMA controller:

- Support independent 8, 16 or 32 bit single value or block transfers
- Supports variable sized queues and circular queues
- Source and destination address registers are independently configured to post-increment or remain constant
- Each transfer is initiated by a peripheral, CPU, periodic timer interrupt or eDMA channel request
- Each eDMA channel can optionally send an interrupt request to the CPU on completion of a single value or block transfer
- DMA transfers possible between system memories, SPIs, I^2C, ADC, UART, eMIOS200 and General Purpose I/Os
- Programmable DMA channel mux allows assignment of any DMA source to any available DMA channel with up to a total of 64 potential request sources.

1.7.17 Crossbar Switch (XBAR)

The Crossbar Switch allows concurrent accesses between masters and slaves, and provides these features:

- Up to 6 master ports
  - Masters: Z6 CPU, Z0 CPU, eDMA, FlexRay, FEC, MLB
- Multiple bus slaves to enable access to flash, SRAM ports and peripherals
- Multiple AIPS bridges to support connection to all peripheral modules
- Crossbar supports consecutive transfers from master to available slaves
- 32-bit internal address bus, 32-bit internal data bus
- User configurable priority arbitration based for masters
- Temporary dynamic priority elevation for IOP and DMA

1.7.18 Memory Protection Unit (MPU)

The MPU provides the following features:

- Supports up to 16 region descriptors for per-master protection
- Start and end address defined with 32-byte granularity
- Overlapping regions supported
- Protection attributes can optionally include process ID
- Protection offered for 4 concurrent read ports
- Read and write attributes for all masters
- Execute and supervisor/user mode attributes for processor masters

NOTE

The MPU is available on the PXN20 only.

1.7.19 Interrupt Controller (INTC)

The PXN20 implements an interrupt controller that features the following:

- Unique 9-bit vector for each of the 316 separate interrupt sources (22 reserved)
- 8 software triggerable interrupt sources
- 16 priority levels with fixed hardware arbitration within priority levels for each interrupt source
- Ability to modify the ISR or task priority.
  - Modifying the priority can be used to implement the priority ceiling protocol for accessing shared resources.
- External high priority interrupt directly accessing the main core critical interrupt mechanism
- Interrupt steering between main CPU and IOP
  - Independent selection of any interrupt source to be routed through IOP
  - Interrupts share same priority level between IOP and CPU

1.7.20 System Clocks and Clock Generation

The following list summarizes the system clock and clock generation on the PXN20:

- System clock can be derived from the following sources
  - 4–40 MHz XTAL
  - FMPLL
  - 16 MHz IRC oscillator
- Programmable output clock divider of system clock (÷1, ÷2, ÷4, ÷8, ÷16)
- Separate programmable peripheral bus clock divider ratio (÷1, ÷2, ÷4, ÷8) applied to system clock
- Frequency Modulated Phase-locked loop (FMPLL)
  - Input clock frequency from 4 MHz to 40 MHz
  - Clock source from external oscillator
  - Lock detect circuitry continuously monitors lock status
  - Loss of clock (LOC) detection for reference and feedback clocks
  - On-chip loop filter (for improved electromagnetic interference performance and reduces number of external components required)
- On-chip crystal oscillatorsupporting 4 MHz to 40 MHz crystals
- Dedicated 16 MHz internal RC oscillator
  - 16 MHz internal RC oscillator supports low speed code execution and clocking of peripherals through selection as the system clock
  - Used as default clock source out of reset
  - Provides a clock for rapid start-up from low power modes
  - Provides a clock for Software Watchdog Timer (SWT)
  - Provides a back-up clock in the event of PLL or external oscillator clock failure
  - 5% accuracy over the operating temperature range (after factory trim)

  - Trimming registers to support frequency adjustment with in-application calibration
- Dedicated internal 128 kHz internal RC oscillator for low power mode operation and self wake-up
  - 5% accuracy (after factory trim)
  - Trimming registers to support improve accuracy with in-application calibration
- Dedicated 32 kHz external oscillator for accurate timed wake-up

1.7.21 System Integration Unit (SIU)

The SIU features the following:

- Up to four levels of internal pin multiplexing, allowing exceptional flexibility in the allocation of device functions for each package
- Centralized general purpose input output (GPIO) control of up to 155 input/output pins (package dependent)
- All GPIO pins can be independently configured to support pull-up, pull down, or no pull
- Reading and writing to GPIO supported both as individual pins and 16-bit wide ports
- The majority of the peripheral pins can be alternatively configured as both general purpose input or output pins. The exception is selected precision ADC channels which support alternative configuration as general purpose inputs only.
- Direct readback of the pin value supported on all digital output pins through the SIU
- Configurable digital input filter that can be applied to up to 32 general purpose input pins for noise elimination on external wakeups

1.7.22 Software Watchdog Timer (SWT)

The Watchdog timer on the PXN20 features the following:

- Watchdog enabled out of reset with default 10 ms timeout from internal 16 MHz IRC clock
- Supports normal and windowed mode
- Support for protected access to watchdog control registers with optional soft and hard locks
  - Soft lock allows the lock to be overridden by writing a special software code
  - Hard lock prevents any changes until after a reset, once enabled
- Watchdog supports optional halting during low power modes
- Configurable response on timeout: reset, interrupt, or interrupt followed by reset

1.7.23 Boot Assist Module (BAM)

The BAM is implemented as follows:

- Configures device to support code download via CAN or UART and execution of download routine
- Multiple bootcode starting locations out of reset through implementation of search for valid reset configuration halfword

1.7.24 Dual-Channel FlexRay Controller (FR)

The dual-channel FlexRay controller features the following:

- Full implementation of FlexRay Protocol Specification 2.1, RevA
- Single channel support
  - FlexRay Port A can be configured to be connected either to physical FlexRay channel A or physical FlexRay channel B
- FlexRay bus data rates of 10, 8, 5, and 2.5 Mbit/s supported
- Up to 128 configurable message buffers with
  - Individual frame ID filtering
  - Individual channel ID filtering
  - Individual cycle counter filtering
- Message buffer header, status and payload data stored in dedicated FlexRay memory
  - Allows for flexible and efficient message buffer implementation
  - Consistent data access ensured by means of buffer locking scheme
  - Application can lock multiple buffers at the same time
- Message buffers can be configured as:
  - Receive message buffer
  - Single buffered transmit message buffer
  - Double buffered transmit message buffer (combines two single buffered message buffers)
- Individual message buffer reconfiguration supported
- Two independent receive FIFOs
  - One receive FIFO per channel
  - Up to 255 entries for each FIFO
  - Global frame ID filtering, based on both value/mask filters and range filters
  - Global channel ID filtering
  - Global message ID filtering for dynamic segment
- Size of message buffer payload data configurable from 0 up to 254 bytes
- Two independent message buffer segments with configurable size of payload data section
  - Each segment can contain message buffers assigned to the static segment and message buffers assigned to the dynamic segment at the same time
- Support for independent internal clock source provided to module from a separate external 40 MHz crystal
- 1 absolute timer
  - 1 timer that can be configured to absolute or relative

1.7.25 Media Local Bus (MLB)

The following summarizes the MLB configuration:

- Support of 16 logical channels running at a maximum speed of 1024 Fs
- Transmission of commands and data and reception of receive status when functioning as the transmitting device associated with a logical channel address
- Reception of commands and data and transmission as receive status responses when functioning as the receiving device associated with a logical channel address
- System channel command handling
- Internal DMA operation
- Local channel buffer RAM (single port RAM) size of 2048 × 36 bits words
- Support for 3-pin only
- Support for MLB I/O voltage specs 2.5 V (nominal) and 3.3 V (nominal)

NOTE

The MLB is available on the PXN21 only.

1.7.26 Real Time Counter (RTC)

Real Time counter supports wake-up from Low Power modes or real time clock generation

- Configurable resolution for different timeout periods
  - 1 sec resolution for > 1 hour period
  - 1 ms resolution for 2-second period
- Selectable clock sources from:
  - Internal 128 kHz RC oscillator
  - Internal 16 MHz RC oscillator
  - 32 kHz external oscillator
- RTC supports continued operation through reset, count only reset manually, or by power on reset (POR)

1.7.27 JTAG Controller (JTAGC)

The JTAGC is compliant with the IEEE 1149.1-2001standard and has the following main features:

- IEEE 1149.1-2001 test access port (TAP) interface
- A JCOMP input that provides the ability to share the TAP
- A 5-bit instruction register that supports several IEEE 1149.1-2001 defined instructions, as well as several public and private MCU specific instructions
- Three test data registers: Bypass register, boundary scan register and a device identification register
- Supporting boundary scan testing
- TAP controller state machine

1.7.28 Nexus Development Interface (NDI)

The NDI module is compliant with the IEEE-ISTO 5001-2003 standard. The following features are implemented, but only available on the 256 MAPBGA emulation package:

- 17-bit full duplex pin interface for medium and high visibility throughput
  - Full port mode (12 MDO)
  - Auxiliary input port (MCKO, 12xMDO, 2xMSEO, EVTO, EVTI)
  - Auxiliary output port
  - 5 pin JTAG port (JCOMP, TDI, TDO, TMS and TCK)

The NPC block performs the following functions

- Controls arbitration between e200Z6 and e200Z0 Nexus modules to the Nexus Auxiliary output port
- Generates full port mode indication output port
- Generates MCKO and frequency division (1/2, 1/4, 1/8).
- Controls sharing of EVTO/EVTI
- Enables gating of MCKO when the auxiliary output port is idle.

e200Z6 development support features (Nexus class3)

- IEEE-ISTO 5001-2003 standard class 3 compliant
- Program trace via branch trace messaging (BTM)
- Data trace via data write messaging (DWM) and data read messaging (DRM). This allows the development tools to trace reads and /or writes to selected internal memory resources
- Ownership via ownership trace messaging (OTM). OTM facilitates ownership trace by providing visibility of which process ID or operating system task is activated
- Run-time access to the e200Z6 memory map via the JTAG port
- Watchpoint messaging
- Watchpoint trigger enable of program and/or data trace messaging

e200Z0 development support features (Nexus class 2+)

- IEEE-ISTO 5001-2003 standard class 2 compliant with additional class 3 and 4 features available
- Program trace via branch trace messaging (BTM)
- Ownership via ownership trace messaging (OTM)
  - OTM facilitates ownership trace by providing visibility of which process ID or operating system task is activated
- Run-time access to the e200Z6 memory map via the JTAG port
- Watchpoint messaging
- Watchpoint trigger enable of program and/or data trace messaging

Capability of an event output signal from either core to generate a debug request in the other core

- All Nexus port pins operate at 3.3 V levels

- Nexus supports debug through reset and low power

1.8 Developer Support

This family of MCUs is supported by Freescale's Tower Development System as well as a broad set of advanced debug and runtime software:

- CodeWarrior
- FreeMaster
- MQX
- RAppID Init
- RAppID Toolbox
- Green Hills

Chapter 2
Memory Map

2.1 Introduction

This section describes the PXN20 memory map.

All addresses in the device, including those that are reserved, are identified in Table 2-1. The addresses represent the physical addresses assigned to each IP block.

Table 2-1. PXN20 System Memory Map

Address		Size (KB)	Region Name	Comments	Unimple-mented on PXS20	Unimple-mented on PXS21
Start	End					
Flash (AXBS Port S0 and S1)						
0x0000_0000	0x0000_3FFF	16	Program/Data Flash	LAS Block L0		
0x0000_4000	0x0000_7FFF	16	Program/Data Flash	LAS Block L1		
0x0000_8000	0x0000_BFFF	16	Program/Data Flash	LAS Block L2		
0x0000_C000	0x0000_FFFF	16	Program/Data Flash	LAS Block L3		
0x0001_0000	0x0001_3FFF	16	Program/Data Flash	LAS Block L4		
0x0001_4000	0x0001_7FFF	16	Program/Data Flash	LAS Block L5		
0x0001_8000	0x0001_BFFF	16	Program/Data Flash	LAS Block L6		
0x0001_C000	0x0001_FFFF	16	Program/Data Flash	LAS Block L7		
0x0002_0000	0x0002_FFFF	64	Program/Data Flash	LAS Block L8		
0x0003_0000	0x0003_FFFF	64	Program/Data Flash	LAS Block L9		
0x0004_0000	0x0005_FFFF	128	Program/Data Flash	MAS Block M0		
0x0006_0000	0x0007_FFFF	128	Program/Data Flash	MAS Block M1		
0x0008_0000	0x000B_FFFF	256	Program/Data Flash	HAS Block H0		
0x000C_0000	0x000F_FFFF	256	Program/Data Flash			
0x0010_0000	0x0013_FFFF	256	Program/Data Flash	HAS Block H1		
0x0014_0000	0x0017_FFFF	256	Program/Data Flash			
0x0018_0000	0x001B_FFFF	256	Program/Data Flash	HAS Block H2		
0x001C_0000	0x001F_FFFF	256	Program/Data Flash			
0x0020_0000	0x00FF_BFFF	14,320	**Reserved**			
0x00FF_C000	0x00FF_FFFF	16	Shadow Row			
0x0100_0000	0x1FFF_FFFF	507,904	Flash Emulation Mapping			
External Bus Interface						

Table 2-1. PXN20 System Memory Map (continued)

Address		Size (KB)	Region Name	Comments	Unimple-mented on PXS20	Unimple-mented on PXS21
Start	**End**					
0x2000_0000	0x3FFF_FFFF	524,288	**Reserved for External Bus Interface**			
SRAM (AXBS Ports S2 and S3)						
0x4000_0000	0x4007_FFFF	512	SRAM (AXBS Port S2)	This 1 MB address space is mirrored 512 times in the address range 0x4000_0000 to 0x5FFF_FFFF		> 128 KB
0x4008_0000	0x4009_3FFF	80	SRAM (AXBS Port S3)			X
0x4009_4000	0x400F_FFFF	432	**Reserved**			
0x4010_0000	0x5FFF_FFFF	523,264	Mirrored address space			
0x6000_0000	0xBFFF_FFFF	1,572,864	**Reserved**			
Peripherals AIPS_A (AXBS Port S6)						
0xC000_0000	0xC3EF_FFFF	64,512	**Reserved**			
0xC3F0_0000	0xC3F0_3FFF	16	**Reserved**			
0xC3F0_4000	0xC3F7_FFFF	496	**Reserved**			
0xC3F8_0000	0xC3F8_3FFF	16	**Reserved**			
0xC3F8_4000	0xC3F8_7FFF	16	MLB_DIM Configuration			X
0xC3F8_8000	0xC3F8_BFFF	16	I^2C_C			
0xC3F8_C000	0xC3F8_FFFF	16	I^2C_D			
0xC3F9_0000	0xC3F9_3FFF	16	DSPI_C			
0xC3F9_4000	0xC3F9_7FFF	16	DSPI_D			
0xC3F9_8000	0xC3F9_BFFF	16	**Reserved**			
0xC3F9_C000	0xC3F9_FFFF	16	**Reserved**			
0xC3FA_0000	0xC3FA_3FFF	16	eSCI_J		X	
0xC3FA_4000	0xC3FA_7FFF	16	eSCI_K		X	
0xC3FA_8000	0xC3FA_BFFF	16	eSCI_L		X	
0xC3FA_C000	0xC3FA_FFFF	16	eSCI_M		X	
0xC3FB_0000	0xC3FB_3FFF	16	**Reserved**			
0xC3FB_4000	0xC3FB_7FFF	16	**Reserved**			
0xC3FB_8000	0xC3FB_BFFF	16	**Reserved**			
0xC3FB_C000	0xC3FB_FFFF	16	**Reserved**			
0xC3FC_0000	0xC3FC_3FFF	16	**Reserved**			
0xC3FC_4000	0xC3FC_7FFF	16	**Reserved**			
0xC3FC_8000	0xC3FC_BFFF	16	**Reserved**			

Table 2-1. PXN20 System Memory Map (continued)

Address		Size (KB)	Region Name	Comments	Unimple-mented on PXS20	Unimple-mented on PXS21
Start	End					
0xC3FC_C000	0xC3FC_FFFF	16	**Reserved**			
0xC3FD_0000	0xC3FD_3FFF	16	**Reserved**			
0xC3FD_4000	0xC3FD_7FFF	16	**Reserved**			
0xC3FD_8000	0xC3FD_BFFF	16	**Reserved**			
0xC3FD_C000	0xC3FD_FFFF	16	FlexRay			X
0xC3FE_0000	0xC3FE_3FFF	16	**Reserved**			
0xC3FE_4000	0xC3FE_7FFF	16	**Reserved**			
0xC3FE_8000	0xC3FF_3FFF	48	**Reserved**			
0xC3FF_4000	0xC3FF_7FFF	16	**Reserved**			
0xC3FF_8000	0xDFFF_FFFF	458,752	**Reserved**			
Peripherals AIPS_B (AXBS Port S7)						
0xE000_0000	0xFFEF_FFFF	523,264	**Reserved**			
0xFFF0_0000	0xFFF0_3FFF	16	**Reserved**			
0xFFF0_4000	0xFFF0_7FFF	16	AXBS			
0xFFF0_8000	0xFFF0_FFFF	32	**Reserved**			
0xFFF1_0000	0xFFF1_3FFF	16	Sema4			
0xFFF1_4000	0xFFF1_7FFF	16	MPU		X	
0xFFF1_8000	0xFFF3_7FFF	128	**Reserved**			
0xFFF3_8000	0xFFF3_BFFF	16	SWT			
0xFFF3_C000	0xFFF3_FFFF	16	STM			
0xFFF4_0000	0xFFF4_3FFF	16	ECSM			
0xFFF4_4000	0xFFF4_7FFF	16	eDMA		Channels 16–31	
0xFFF4_8000	0xFFF4_BFFF	16	INTC			
0xFFF4_C000	0xFFF4_FFFF	16	FEC			X
0xFFF5_0000	0xFFF7_FFFF	192	**Reserved**			
0xFFF8_0000	0xFFF8_3FFF	16	ADC_A			
0xFFF8_4000	0xFFF8_7FFF	16	**Reserved**			
0xFFF8_8000	0xFFF8_BFFF	16	I^2C_A			
0xFFF8_C000	0xFFF8_FFFF	16	I^2C_B			
0xFFF9_0000	0xFFF9_3FFF	16	DSPI_A			
0xFFF9_4000	0xFFF9_7FFF	16	DSPI_B			

Table 2-1. PXN20 System Memory Map (continued)

Address		Size (KB)	Region Name	Comments	Unimple-mented on PXS20	Unimple-mented on PXS21
Start	End					
0xFFF9_8000	0xFFF9_BFFF	16	**Reserved**			
0xFFF9C000	0xFFF9_FFFF	16	**Reserved**			
0xFFFA_0000	0xFFFA_3FFF	16	eSCI_A			
0xFFFA_4000	0xFFFA_7FFF	16	eSCI_B			
0xFFFA_8000	0xFFFA_BFFF	16	eSCI_C			
0xFFFA_C000	0xFFFA_FFFF	16	eSCI_D			
0xFFFB_0000	0xFFFB_3FFF	16	eSCI_E			
0xFFFB_4000	0xFFFB_7FFF	16	eSCI_F			
0xFFFB_8000	0xFFFB_BFFF	16	eSCI_G		X	
0xFFFB_C000	0xFFFB_FFFF	16	eSCI_H		X	
0xFFFC_0000	0xFFFC_3FFF	16	FlexCan_A			
0xFFFC_4000	0xFFFC_7FFF	16	FlexCan_B			
0xFFFC_8000	0xFFFC_BFFF	16	FlexCan_C			
0xFFFC_C000	0xFFFC_FFFF	16	FlexCan_D			
0xFFFD_0000	0xFFFD_3FFF	16	FlexCan_E			
0xFFFD_4000	0xFFFD_7FFF	16	FlexCan_F			X
0xFFFD_8000	0xFFFD_BFFF	16	CTU_A		X	
0xFFFD_C000	0xFFFD_FFFF	16	DMA Multiplexer			
0xFFFE_0000	0xFFFE_3FFF	16	PIT			
0xFFFE_4000	0xFFFE_7FFF	16	eMIOS_A		Channels 24–31	
0xFFFE_8000	0xFFFE_BFFF	16	SIU			
0xFFFE_C000	0xFFFE_FFFF	16	CRP			
0xFFFF_0000	0xFFFF_3FFF	16	PLL			
0xFFFF_4000	0xFFFF_7FFF	16	**Reserved**			
0xFFFF_8000	0xFFFF_BFFF	16	PFlash Configuration			
0xFFFF_C000	0xFFFF_FFFF	16	BAM (upper 8K)			

Chapter 3
Signal Description

3.1 Introduction

This chapter describes signals that connect off-chip. It includes a signal properties summary, power and ground segmentation summary, package pinouts, and detailed descriptions of signals. Because the PXN20 comes in multiple packages, some signals may not be available on every package. Refer to the *PXN20 Microcontroller Data Sheet* for electrical characteristics.

3.2 Signal Properties Summary

Table 3-1 shows the signals properties for each pin on PXN20. For all port pins that have an associated SIU_PCR*n* register to control pin properties, the supported functions column lists the functions associated with the programming of the SIU_PCR*n*[PA] bit in the order: general-purpose input/output (GPIO), function 1, function 2, and function 3 (see Figure 3-1). When an alternate function is not implemented for a value of SIU_PCR*n*[PA], a dash is shown in the Description column and the respective value in the PA bitfield is reserved.

Supported Functions[2]	GPIO (PCR) Num[3]	PA[4]	Description
PA[0] AN[0]	0	00 01 10 11	Port A GPI ADC Analog Input — —

GPIO
Function 1
Functions 2 and 3 not implemented

Figure 3-1. Supported Functions Example

Table 3-1. PXN20 Signal Properties

Pin Name[1]	Supported Functions[2]	GPIO (PCR) Num[3]	PA[4]	Description	I/O Type	Volt-age	Pad Type[5]	Status		Package Pin Locations
								During Reset[6]	After Reset[7]	208 BGA
Port A (16)										
PA0	PA[0] AN[0]	0	00 01 10 11	Port A GPI ADC Analog Input — —	I I — —	V_{DDA}	IHA	—	—	D15
PA1	PA[1] AN[1]	1	00 01 10 11	Port A GPI ADC Analog Input — —	I I — —	V_{DDA}	IHA	—	—	E15
PA2	PA[2] AN[2]	2	00 01 10 11	Port A GPI ADC Analog Input — —	I I — —	V_{DDA}	IHA	—	—	F16
PA3	PA[3] AN[3]	3	00 01 10 11	Port A GPI ADC Analog Input — —	I I — —	V_{DDA}	IHA	—	—	F15
PA4	PA[4] AN[4]	4	00 01 10 11	Port A GPI ADC Analog Input — —	I I — —	V_{DDA}	IHA	—	—	G16
PA5	PA[5] AN[5]	5	00 01 10 11	Port A GPI ADC Analog Input — —	I I — —	V_{DDA}	IHA	—	—	G15
PA6	PA[6] AN[6]	6	00 01 10 11	Port A GPI ADC Analog Input — —	I I — —	V_{DDA}	IHA	—	—	H16
PA7	PA[7] AN[7]	7	00 01 10 11	Port A GPI ADC Analog Input — —	I I — —	V_{DDA}	IHA	—	—	G14
PA8	PA[8] AN[8]	8	00 01 10 11	Port A GPI ADC Analog Input — —	I I — —	V_{DDA}	IHA	—	—	F14
PA9	PA[9] AN[9]	9	00 01 10 11	Port A GPI ADC Analog Input — —	I I — —	V_{DDA}	IHA	—	—	E14

Table 3-1. PXN20 Signal Properties (continued)

Pin Name[1]	Supported Functions[2]	GPIO (PCR) Num[3]	PA[4]	Description	I/O Type	Volt-age	Pad Type[5]	Status		Package Pin Locations
								During Reset[6]	After Reset[7]	208 BGA
PA10	PA[10] AN[10]	10	00 01 10 11	Port A GPI ADC Analog Input — —	I I — —	V_{DDA}	IHA	—	—	D13
PA11	PA[11] AN[11]	11	00 01 10 11	Port A GPI ADC Analog Input — —	I I — —	V_{DDA}	IHA	—	—	E13
PA12	PA[12] AN[12]	12	00 01 10 11	Port A GPI ADC Analog Input — —	I I — —	V_{DDA}	IHA	—	—	D14
PA13	PA[13] AN[13]	13	00 01 10 11	Port A GPI ADC Analog Input — —	I I — —	V_{DDA}	IHA	—	—	F13
PA14	PA[14] AN[14] EXTAL32	14	00 01 10 11	Port A GPI ADC Analog Input External 32 kHz Crystal In —	I I I —	V_{DDA}	IHA	—	—	D16
PA15	PA[15] AN[15] XTAL32	15	00 01 10 11	Port A GPI ADC Analog Input External 32 kHz Crystal Out —	I I O —	V_{DDA}	IHA	—	—	E16
Port B (16)										
PB0	PB[0] AN[16]/ANW	16	00 01 10 11	Port B GPIO ADC Analog Input/Mux In — —	I/O I — —	V_{DDE1}	SHA	—	—	B14
PB1	PB[1] AN[17]/ANX	17	00 01 10 11	Port B GPIO ADC Analog Input/Mux In — —	I/O I — —	V_{DDE1}	SHA	—	—	C14
PB2	PB[2] AN[18]/ANY	18	00 01 10 11	Port B GPIO ADC Analog Input/Mux In — —	I/O I — —	V_{DDE1}	SHA	—	—	B13
PB3	PB[3] AN[19]/ANZ	19	00 01 10 11	Port B GPIO ADC Analog Input/Mux In — —	I/O I — —	V_{DDE1}	SHA	—	—	C13

Table 3-1. PXN20 Signal Properties (continued)

Pin Name[1]	Supported Functions[2]	GPIO (PCR) Num[3]	PA[4]	Description	I/O Type	Volt-age	Pad Type[5]	Status		Package Pin Locations
								During Reset[6]	After Reset[7]	208 BGA
PB4	PB[4] AN[20]	20	00 01 10 11	Port B GPIO ADC Analog Input — —	I/O I — —	V_{DDE1}	SHA	—	—	C12
PB5	PB[5] AN[21]	21	00 01 10 11	Port B GPIO ADC Analog Input — —	I/O I — —	V_{DDE1}	SHA	—	—	D12
PB6	PB[6] AN[22]	22	00 01 10 11	Port B GPIO ADC Analog Input — —	I/O I — —	V_{DDE1}	SHA	—	—	C11
PB7	PB[7] AN[23]	23	00 01 10 11	Port B GPIO ADC Analog Input — —	I/O I — —	V_{DDE1}	SHA	—	—	D11
PB8	PB[8] AN[24] PCS_A[2]	24	00 01 10 11	Port B GPIO ADC Analog Input DSPI_A Peripheral Chip Select —	I/O I O —	V_{DDE1}	SHA	—	—	A10
PB9	PB[9] AN[25] PCS_A[3]	25	00 01 10 11	Port B GPIO ADC Analog Input DSPI_A Peripheral Chip Select —	I/O I O —	V_{DDE1}	SHA	—	—	B12
PB10	PB[10] AN[26] PCS_B[4]	26	00 01 10 11	Port B GPIO ADC Analog Input DSPI_B Peripheral Chip Select —	I/O I O —	V_{DDE1}	SHA	—	—	A9
PB11	PB[11] AN[27] PCS_B[5]	27	00 01 10 11	Port B GPIO ADC Analog Input DSPI_B Peripheral Chip Select —	I/O I O —	V_{DDE1}	SHA	—	—	B9
PB12	PB[12] AN[28] PCS_C[1]	28	00 01 10 11	Port B GPIO ADC Analog Input DSPI_C Peripheral Chip Select —	I/O I O —	V_{DDE1}	SHA	—	—	C10
PB13	PB[13] AN[29] PCS_C[2]	29	00 01 10 11	Port B GPIO ADC Analog Input DSPI_C Peripheral Chip Select —	I/O I O —	V_{DDE1}	SHA	—	—	A8
PB14	PB[14] AN[30] PCS_D[3]	30	00 01 10 11	Port B GPIO ADC Analog Input DSPI_D Peripheral Chip Select —	I/O I O —	V_{DDE1}	SHA	—	—	B8

Table 3-1. PXN20 Signal Properties (continued)

Pin Name[1]	Supported Functions[2]	GPIO (PCR) Num[3]	PA[4]	Description	I/O Type	Volt-age	Pad Type[5]	Status		Package Pin Locations
								During Reset[6]	After Reset[7]	208 BGA
PB15	PB[15] AN[31] PCS_D[4]	31	00 01 10 11	Port B GPIO ADC Analog Input DSPI_D Peripheral Chip Select —	I/O I O —	V_{DDE1}	SHA	—	—	C9
Port C (16)										
PC0	PC[0] AN[32]	32	00 01 10 11	Port C GPIO ADC Analog Input — —	I/O I — —	V_{DDE1}	SHA	—	—	D9
PC1	PC[1] AN[33]	33	00 01 10 11	Port C GPIO ADC Analog Input — —	I/O I — —	V_{DDE1}	SHA	—	—	C8
PC2	PC[2] AN[34] EVTI	34	00 01 10 11	Port C GPIO ADC Analog Input Nexus Event In —	I/O I I —	V_{DDE1}	SHA	—	—	A7
PC3	PC[3] AN[35] EVTO	35	00 01 10 11	Port C GPIO ADC Analog Input Nexus Event Out —	I/O I O —	V_{DDE1}	SHA	—	—	B7
PC4	PC[4] AN[36]	36	00 01 10 11	Port C GPIO ADC Analog Input — —	I/O I — —	V_{DDE1}	SHA	—	—	D8
PC5	PC[5] AN[37] Z6NMI	37	00 01 10 11	Port C GPIO ADC Analog Input Z6 Core Non-Maskable Interrupt —	I/O I I —	V_{DDE1}	SHA	—	—	C6
PC6	PC[6] AN[38] Z0NMI	38	00 01 10 11	Port C GPIO ADC Analog Input Z0 Core Non-Maskable Interrupt —	I/O I I —	V_{DDE1}	SHA	—	—	C7
PC7	PC[7] AN[39] FR_DBG3	39	00 01 10 11	Port C GPIO ADC Analog Input FlexRay Debug —	I/O I O —	V_{DDE1}	SHA	—	—	A6
PC8	PC[8] AN[40] FR_DBG2	40	00 01 10 11	Port C GPIO ADC Analog Input FlexRay Debug —	I/O I O —	V_{DDE1}	SHA	—	—	B6

Table 3-1. PXN20 Signal Properties (continued)

Pin Name[1]	Supported Functions[2]	GPIO (PCR) Num[3]	PA[4]	Description	I/O Type	Volt-age	Pad Type[5]	Status		Package Pin Locations
								During Reset[6]	After Reset[7]	208 BGA
PC9	PC[9] AN[41] FR_DBG1	41	00 01 10 11	Port C GPIO ADC Analog Input FlexRay Debug —	I/O I O —	V_{DDE1}	SHA	—	—	A5
PC10	PC[10] AN[42] FR_DBG0	42	00 01 10 11	Port C GPIO ADC Analog Input FlexRay Debug —	I/O I O —	V_{DDE1}	SHA	—	—	B5
PC11	PC[11] AN[43] SCL_C	43	00 01 10 11	Port C GPIO ADC Analog Input I^2C_C Serial Clock —	I/O I I/O —	V_{DDE1}	SHA	—	—	B4
PC12	PC[12] AN[44] SDA_C	44	00 01 10 11	Port C GPIO ADC Analog Input I^2C_C Serial Data —	I/O I I/O —	V_{DDE1}	SHA	—	—	A4
PC13	PC[13] AN[45] — MA[0]	45	00 01 10 11	Port C GPIO ADC Analog Input — ADC Ext. Mux Address Select	I/O I — O	V_{DDE1}	SHA	—	—	C5
PC14	PC[14] AN[46] MA[1]	46	00 01 10 11	Port C GPIO ADC Analog Input ADC Ext. Mux Address Select —	I/O I — O	V_{DDE1}	SHA	—	—	C4
PC15	PC[15] AN[47] MA[2]	47	00 01 10 11	Port C GPIO ADC Analog Input ADC Ext. Mux Address Select —	I/O I O —	V_{DDE1}	SHA	—	—	D5
Port D (16)										
PD0	PD[0] CNTX_A	48	00 01 10 11	Port D GPIO FlexCAN_A Transmit — —	I/O O — —	V_{DDE2}	SH	—	—	A2
PD1	PD[1] CNRX_A	49	00 01 10 11	Port D GPIO FlexCAN_A Receive — —	I/O I — —	V_{DDE2}	SH	—	—	B2
PD2	PD[2] CNTX_B	50	00 01 10 11	Port D GPIO FlexCAN_B Transmit — —	I/O O — —	V_{DDE2}	SH	—	—	B1

Table 3-1. PXN20 Signal Properties (continued)

Pin Name[1]	Supported Functions[2]	GPIO (PCR) Num[3]	PA[4]	Description	I/O Type	Volt-age	Pad Type[5]	Status		Package Pin Locations
								During Reset[6]	After Reset[7]	208 BGA
PD3	PD[3] CNRX_B	51	00 01 10 11	Port D GPIO FlexCAN_B Receive — —	I/O I — —	V_{DDE2}	SH	—	—	C1
PD4	PD[4] CNTX_C	52	00 01 10 11	Port D GPIO FlexCAN_C Transmit — —	I/O O — —	V_{DDE2}	SH	—	—	C2
PD5	PD[5] CNRX_C	53	00 01 10 11	Port D GPIO FlexCAN_C Receive — —	I/O I — —	V_{DDE2}	SH	—	—	D1
PD6	PD[6] CNTX_D TXD_K SCL_B	54	00 01 10 11	Port D GPIO FlexCAN_D Transmit SCI_K Transmit I^2C_B Serial Clock	I/O O O I/O	V_{DDE2}	SH	—	—	D2
PD7	PD[7] CNRX_D RXD_K SDA_B	55	00 01 10 11	Port D GPIO FlexCAN_D Receive SCI_K Receive I^2C_B Serial Data	I/O I I I/O	V_{DDE2}	SH	—	—	E1
PD8	PD[8] CNTX_E TXD_L SCL_C	56	00 01 10 11	Port D GPIO FlexCAN_E Transmit SCI_L Transmit I^2C_C Serial Clock	I/O O O I/O	V_{DDE2}	SH	—	—	E2
PD9	PD[9] CNRX_E RXD_L SDA_C	57	00 01 10 11	Port D GPIO FlexCAN_E Receive SCI_L Receive I^2C_C Serial Data	I/O I I I/O	V_{DDE2}	SH	—	—	F1
PD10	PD[10] CNTX_F TXD_M SCL_D	58	00 01 10 11	Port D GPIO FlexCAN_F Transmit SCI_M Transmit I^2C_D Serial Clock	I/O O O I/O	V_{DDE2}	SH	—	—	F2
PD11	PD[11] CNRX_F RXD_M SDA_D	59	00 01 10 11	Port D GPIO FlexCAN_F Receive SCI_M Receive I^2C_D Serial Data	I/O I I I/O	V_{DDE2}	SH	—	—	G1
PD12	PD[12] TXD_A	60	00 01 10 11	Port D GPIO eSCI_A Transmit — —	I/O O — —	V_{DDE2}	SH	—	—	G2
PD13	PD[13] RXD_A	61	00 01 10 11	Port D GPIO eSCI_A Receive — —	I/O I — —	V_{DDE2}	SH	—	—	H1

Table 3-1. PXN20 Signal Properties (continued)

Pin Name[1]	Supported Functions[2]	GPIO (PCR) Num[3]	PA[4]	Description	I/O Type	Volt-age	Pad Type[5]	Status		Package Pin Locations
								During Reset[6]	After Reset[7]	208 BGA
PD14	PD[14] TXD_B	62	00 01 10 11	Port D GPIO eSCI_B Transmit — —	I/O O — —	V_{DDE2}	SH	—	—	C3
PD15	PD[15] RXD_B	63	00 01 10 11	Port D GPIO eSCI_B Receive — —	I/O I — —	V_{DDE2}	SH	—	—	D3
Port E (16)										
PE0	PE[0] TXD_C eMIOS[31]	64	00 01 10 11	Port E GPIO eSCI_C Transmit eMIOS Channel —	I/O O I/O —	V_{DDE2}	SH	—	—	E3
PE1	PE[1] RXD_C eMIOS[30]	65	00 01 10 11	Port E GPIO eSCI_C Receive eMIOS Channel —	I/O I I/O	V_{DDE2}	SH	—	—	E4
PE2	PE[2] TXD_D eMIOS[29]	66	00 01 10 11	Port E GPIO eSCI_D Transmit eMIOS Channel —	I/O O I/O	V_{DDE2}	SH	—	—	F4
PE3	PE[3] RXD_D eMIOS[28]	67	00 01 10 11	Port E GPIO eSCI_D Receive eMIOS Channel —	I/O I I/O	V_{DDE2}	SH	—	—	F3
PE4	PE[4] TXD_E eMIOS[27]	68	00 01 10 11	Port E GPIO eSCI_E Transmit eMIOS Channel —	I/O O I/O	V_{DDE2}	SH	—	—	G3
PE5	PE[5] RXD_E eMIOS[26]	69	00 01 10 11	Port E GPIO eSCI_E Receive eMIOS Channel —	I/O I I/O	V_{DDE2}	SH	—	—	H3
PE6	PE[6] TXD_F eMIOS[25]	70	00 01 10 11	Port E GPIO eSCI_F Transmit eMIOS Channel —	I/O O I/O	V_{DDE2}	SH	—	—	M2
PE7	PE[7] RXD_F eMIOS[24]	71	00 01 10 11	Port E GPIO eSCI_F Receive eMIOS Channel —	I/O I I/O	V_{DDE2}	SH	—	—	L2

Table 3-1. PXN20 Signal Properties (continued)

Pin Name[1]	Supported Functions[2]	GPIO (PCR) Num[3]	PA[4]	Description	I/O Type	Volt-age	Pad Type[5]	Status		Package Pin Locations
								During Reset[6]	After Reset[7]	208 BGA
PE8	PE[8] TXD_G PCS_A[1]	72	00 01 10 11	Port E GPIO eSCI_G Transmit DSPI_A Peripheral Chip Select —	I/O O O	V_{DDE2}	SH	—	—	J4
PE9	PE[9] RXD_G PCS_A[4]	73	00 01 10 11	Port E GPIO eSCI_G Receive DSPI_A Peripheral Chip Select —	I/O I O	V_{DDE2}	SH	—	—	M4
PE10	PE[10] TXD_H PCS_B[3]	74	00 01 10 11	Port E GPIO eSCI_H Transmit DSPI_B Peripheral Chip Select —	I/O O O	V_{DDE2}	SH	—	—	N3
PE11	PE[11] RXD_H PCS_B[2]	75	00 01 10 11	Port E GPIO eSCI_H Receive DSPI_B Peripheral Chip Select —	I/O I O	V_{DDE2}	SH	—	—	N4
PE12	PE[12] TXD_J PCS_C[5]	76	00 01 10 11	Port E GPIO eSCI_J Transmit DSPI_C Peripheral Chip Select —	I/O O O	V_{DDE2}	SH	—	—	P4
PE13	PE[13] RXD_J PCS_C[3]	77	00 01 10 11	Port E GPIO eSCI_J Receive DSPI_C Peripheral Chip Select —	I/O I O	V_{DDE2}	SH	—	—	P5
PE14	PE[14] SCL_A PCS_D[2]	78	00 01 10 11	Port E GPIO I^2C_A Serial Clock DSPI_D Peripheral Chip Select —	I/O I/O O —	V_{DDE2}	SH	—	—	N7
PE15	PE[15] SDA_A PCS_D[5]	79	00 01 10 11	Port E GPIO I^2C_A Serial Data DSPI_D Peripheral Chip Select —	I/O I/O O —	V_{DDE2}	SH	—	—	N6
Port F (16)										
PF0	PF[0] SCK_A	80	00 01 10 11	Port F GPIO DSPI_A Serial Clock — —	I/O I/O — —	V_{DDE2}	MH	—	—	H2
PF1	PF[1] SOUT_A	81	00 01 10 11	Port F GPIO DSPI_A Serial Data Out — —	I/O O — —	V_{DDE2}	MH	—	—	J1

Table 3-1. PXN20 Signal Properties (continued)

Pin Name[1]	Supported Functions[2]	GPIO (PCR) Num[3]	PA[4]	Description	I/O Type	Volt-age	Pad Type[5]	Status		Package Pin Locations
								During Reset[6]	After Reset[7]	208 BGA
PF2	PF[2] SIN_A	82	00 01 10 11	Port F GPIO DSPI_A Serial Data In — —	I/O I — —	V_{DDE2}	SH	—	—	J2
PF3	PF[3] PCS_A[0] PCS_B[5] PCS_C[4]	83	00 01 10 11	Port F GPIO DSPI_A Peripheral Chip Select DSPI_B Peripheral Chip Select DSPI_C Peripheral Chip Select	I/O I/O O O	V_{DDE2}	SH	—	—	N2
PF4	PF[4] SCK_B PCS_A[1] PCS_C[2]	84	00 01 10 11	Port F GPIO DSPI_B Serial Clock DSPI_A Peripheral Chip Select DSPI_C Peripheral Chip Select	I/O I/O O O	V_{DDE2}	MH	—	—	M1
PF5	PF[5] SOUT_B PCS_A[2] PCS_C[3]	85	00 01 10 11	Port F GPIO DSPI_B Serial Data Out DSPI_A Peripheral Chip Select DSPI_C Peripheral Chip Select	I/O O O O	V_{DDE2}	MH	—	—	P2
PF6	PF[6] SIN_B PCS_A[3] PCS_C[5]	86	00 01 10 11	Port F GPIO DSPI_B Serial Data In DSPI_A Peripheral Chip Select DSPI_C Peripheral Chip Select	I/O I O O	V_{DDE2}	SH	—	—	N1
PF7	PF[7] PCS_B[0] PCS_C[5] PCS_D[4]	87	00 01 10 11	Port F GPIO DSPI_B Peripheral Chip Select DSPI_C Peripheral Chip Select DSPI_D Peripheral Chip Select	I/O I/O O O	V_{DDE2}	SH	—	—	R2
PF8	PF[8] SCK_C	88	00 01 10 11	Port F GPIO DSPI_C Serial Clock — —	I/O I/O — —	V_{DDE2}	MH	—	—	P1
PF9	PF[9] SOUT_C	89	00 01 10 11	Port F GPIO DSPI_C Serial Data Out — —	I/O O — —	V_{DDE2}	MH	—	—	T2
PF10	PF[10] SIN_C	90	00 01 10 11	Port F GPIO DSPI_C Serial Data In — —	I/O I — —	V_{DDE2}	SH	—	—	R1
PF11	PF[11] PCS_C[0] PCS_D[5] PCS_A[4]	91	00 01 10 11	Port F GPIO DSPI_C Peripheral Chip Select DSPI_D Peripheral Chip Select DSPI_A Peripheral Chip Select	I/O I/O O O	V_{DDE2}	SH	—	—	R3
PF12	PF[12] SCK_D	92	00 01 10 11	Port F GPIO DSPI_D Serial Clock — —	I/O I/O — —	V_{DDE3}	MH	—	—	N14

Table 3-1. PXN20 Signal Properties (continued)

Pin Name[1]	Supported Functions[2]	GPIO (PCR) Num[3]	PA[4]	Description	I/O Type	Volt-age	Pad Type[5]	Status		Package Pin Locations
								During Reset[6]	After Reset[7]	208 BGA
PF13	PF[13] SOUT_D	93	00 01 10 11	Port F GPIO DSPI_D Serial Data Out — —	I/O O — —	V_{DDE3}	MH	—	—	M14
PF14	PF[14] SIN_D	94	00 01 10 11	Port F GPIO DSPI_D Serial Data In — —	I/O I — —	V_{DDE3}	SH	—	—	P14
PF15	PF[15] PCS_D[0] PCS_A[5] PCS_B[4]	95	00 01 10 11	Port F GPIO DSPI_D Peripheral Chip Select DSPI_A Peripheral Chip Select DSPI_B Peripheral Chip Select	I/O I/O O O	V_{DDE3}	SH	—	—	P13
Port G (16)										
PG0	PG[0] PCS_A[4] PCS_B[3] AN[48]	96	00 01 10 11	Port G GPIO DSPI_A Peripheral Chip Select DSPI_B Peripheral Chip Select ADC Analog Input	I/O O O I	V_{DDE2}	SHA	—	—	B3
PG1	PG[1] PCS_A[5] PCS_B[4] AN[49]	97	00 01 10 11	Port G GPIO DSPI_A Peripheral Chip Select DSPI_B Peripheral Chip Select ADC Analog Input	I/O O O I	V_{DDE2}	SHA	—	—	A3
PG2	PG[2] PCS_D[1] SCL_C AN[50]	98	00 01 10 11	Port G GPIO DSPI_D Peripheral Chip Select I^2C_C Serial Clock ADC Analog Input	I/O O I/O I	V_{DDE3}	SHA	—	—	H14
PG3	PG[3] PCS_D[2] SDA_C AN[51]	99	00 01 10 11	Port G GPIO DSPI_D Peripheral Chip Select I^2C_C Serial Data ADC Analog Input	I/O O I/O I	V_{DDE3}	SHA	—	—	J14
PG4	PG[4] PCS_D[3] SCL_B AN[52]	100	00 01 10 11	Port G GPIO DSPI_D Peripheral Chip Select I^2C_B Serial Clock ADC Analog Input	I/O O I/O I	V_{DDE3}	SHA	—	—	K14
PG5	PG[5] PCS_D[4] SDA_B AN[53]	101	00 01 10 11	Port G GPIO DSPI_D Peripheral Chip Select I^2C_B Serial Data ADC Analog Input	I/O O I/O I	V_{DDE3}	SHA	—	—	L14
PG6	PG[6] PCS_C[1] FEC_MDC AN[54]	102	00 01 10 11	Port G GPIO DSPI_C Peripheral Chip Select Ethernet Mgmt. Data Clock ADC Analog Input	I/O O O I	V_{DDE3}	MHA	—	—	H15

Table 3-1. PXN20 Signal Properties (continued)

Pin Name[1]	Supported Functions[2]	GPIO (PCR) Num[3]	PA[4]	Description	I/O Type	Volt-age	Pad Type[5]	Status		Package Pin Locations
								During Reset[6]	After Reset[7]	208 BGA
PG7	PG[7] PCS_C[2] FEC_MDIO AN[55]	103	00 01 10 11	Port G GPIO DSPI_C Peripheral Chip Select Ethernet Mgmt. Data I/O ADC Analog Input	I/O O I/O I	V_{DDE3}	MHA	—	—	J15
PG8	PG[8] eMIOS[7] FEC_TX_CLK AN[56]	104	00 01 10 11	Port G GPIO eMIOS Channel Ethernet Transmit Clock ADC Analog Input	I/O I/O I I	V_{DDE3}	SHA	—	—	K15
PG9	PG[9] eMIOS[6] FEC_CRS AN[57]	105	00 01 10 11	Port G GPIO eMIOS Channel Ethernet Carrier Sense ADC Analog Input	I/O I/O I I	V_{DDE3}	SHA	—	—	L15
PG10	PG[10] eMIOS[5] FEC_TX_ER AN[58]	106	00 01 10 11	Port G GPIO eMIOS Channel Ethernet Transmit Error ADC Analog Input	I/O I/O O I	V_{DDE3}	MHA	—	—	M15
PG11	PG[11] eMIOS[4] FEC_RX_CLK AN[59]	107	00 01 10 11	Port G GPIO eMIOS Channel Ethernet Receive Clock ADC Analog Input	I/O I/O I I	V_{DDE3}	SHA	—	—	J16
PG12	PG[12] eMIOS[3] FEC_TXD[0] AN[60]	108	00 01 10 11	Port G GPIO eMIOS Channel Ethernet Transmit Data ADC Analog Input	I/O I/O O I	V_{DDE3}	MHA	—	—	K16
PG13	PG[13] eMIOS[2] FEC_TXD[1] AN[61]	109	00 01 10 11	Port G GPIO eMIOS Channel Ethernet Transmit Data ADC Analog Input	I/O I/O O I	V_{DDE3}	MHA	—	—	L16
PG14	PG[14] eMIOS[1] FEC_TXD[2] AN[62]	110	00 01 10 11	Port G GPIO eMIOS Channel Ethernet Transmit Data ADC Analog Input	I/O I/O O I	V_{DDE3}	MHA	—	—	M16
PG15	PG[15] eMIOS[0] FEC_TXD[3] AN[63]	111	00 01 10 11	Port G GPIO eMIOS Channel Ethernet Transmit Data ADC Analog Input	I/O I/O O I	V_{DDE3}	MHA	—	—	N16
Port H (16)										
PH0	PH[0] eMIOS[31] FEC_COL	112	00 01 10 11	Port H GPIO eMIOS Channel Ethernet Collision —	I/O I/O I —	V_{DDE3}	SH	—	—	T14

Table 3-1. PXN20 Signal Properties (continued)

Pin Name[1]	Supported Functions[2]	GPIO (PCR) Num[3]	PA[4]	Description	I/O Type	Volt-age	Pad Type[5]	Status		Package Pin Locations
								During Reset[6]	After Reset[7]	208 BGA
PH1	PH[1] eMIOS[30] FEC_RX_DV	113	00 01 10 11	Port H GPIO eMIOS Channel Ethernet Receive Data Valid —	I/O I/O I —	V_{DDE3}	SH	—	—	P16
PH2	PH[2] eMIOS[29] FEC_TX_EN	114	00 01 10 11	Port H GPIO eMIOS Channel Ethernet Transmit Enable —	I/O I/O O —	V_{DDE3}	MH	—	—	R16
PH3	PH[3] eMIOS[28] FEC_RX_ER	115	00 01 10 11	Port H GPIO eMIOS Channel Ethernet Receive Error —	I/O I/O I —	V_{DDE3}	SH	—	—	N15
PH4	PH[4] eMIOS[27] FEC_RXD[0]	116	00 01 10 11	Port H GPIO eMIOS Channel Ethernet Receive Data —	I/O I/O I —	V_{DDE3}	SH	—	—	P15
PH5	PH[5] eMIOS[26] FEC_RXD[1]	117	00 01 10 11	Port H GPIO eMIOS Channel Ethernet Receive Data —	I/O I/O I —	V_{DDE3}	SH	—	—	R14
PH6	PH[6] eMIOS[25] FEC_RXD[2]	118	00 01 10 11	Port H GPIO eMIOS Channel Ethernet Receive Data —	I/O I/O I —	V_{DDE3}	SH	—	—	R15
PH7	PH[7] eMIOS[24] FEC_RXD[3]	119	00 01 10 11	Port H GPIO eMIOS Channel Ethernet Receive Data —	I/O I/O I —	V_{DDE3}	SH	—	—	T15
PH8	PH[8] eMIOS[23]	120	00 01 10 11	Port H GPIO eMIOS Channel — —	I/O I/O — —	V_{DDE4}	SH	—	—	P7
PH9	PH[9] eMIOS[22]	121	00 01 10 11	Port H GPIO eMIOS Channel — —	I/O I/O — —	V_{DDE4}	SH	—	—	N8
PH10	PH[10] eMIOS[21]	122	00 01 10 11	Port H GPIO eMIOS Channel — —	I/O I/O — —	V_{DDE4}	SH	—	—	P8
PH11	PH[11] eMIOS[20]	123	00 01 10 11	Port H GPIO eMIOS Channel — —	I/O I/O — —	V_{DDE4}	SH	—	—	N9

Table 3-1. PXN20 Signal Properties (continued)

Pin Name[1]	Supported Functions[2]	GPIO (PCR) Num[3]	PA[4]	Description	I/O Type	Volt-age	Pad Type[5]	Status		Package Pin Locations
								During Reset[6]	After Reset[7]	208 BGA
PH12	PH[12] eMIOS[19]	124	00 01 10 11	Port H GPIO eMIOS Channel — —	I/O I/O — —	V_{DDE4}	SH	—	—	P9
PH13	PH[13] eMIOS[18]	125	00 01 10 11	Port H GPIO eMIOS Channel — —	I/O I/O — —	V_{DDE4}	SH	—	—	P10
PH14	PH[14] eMIOS[17]	126	00 01 10 11	Port H GPIO eMIOS Channel — —	I/O I/O — —	V_{DDE4}	SH	—	—	P11
PH15	PH[15] eMIOS[16]	127	00 01 10 11	Port H GPIO eMIOS Channel — —	I/O I/O — —	V_{DDE4}	SH	—	—	N11
Port J (16)										
PJ0	PJ[0] eMIOS[15] PCS_A[4]	128	00 01 10 11	Port J GPIO eMIOS Channel DSPI_A Peripheral Chip Select —	I/O I/O O —	V_{DDE4}	SH	—	—	R7
PJ1	PJ[1] eMIOS[14] PCS_A[5]	129	00 01 10 11	Port J GPIO eMIOS Channel DSPI_A Peripheral Chip Select —	I/O I/O O —	V_{DDE4}	SH	—	—	T7
PJ2	PJ[2] eMIOS[13] PCS_B[1]	130	00 01 10 11	Port J GPIO eMIOS Channel DSPI_B Peripheral Chip Select —	I/O I/O O —	V_{DDE4}	SH	—	—	R8
PJ3	PJ[3] eMIOS[12] PCS_B[2]	131	00 01 10 11	Port J GPIO eMIOS Channel DSPI_B Peripheral Chip Select —	I/O I/O O —	V_{DDE4}	SH	—	—	T8
PJ4	PJ[4] eMIOS[11] PCS_C[3]	132	00 01 10 11	Port J GPIO eMIOS Channel DSPI_C Peripheral Chip Select —	I/O I/O O —	V_{DDE4}	SH	—	—	R9
PJ5	PJ[5] eMIOS[10] PCS_C[4]	133	00 01 10 11	Port J GPIO eMIOS Channel DSPI_C Peripheral Chip Select —	I/O I/O O —	V_{DDE4}	SH	—	—	T9

Table 3-1. PXN20 Signal Properties (continued)

Pin Name[1]	Supported Functions[2]	GPIO (PCR) Num[3]	PA[4]	Description	I/O Type	Volt-age	Pad Type[5]	Status		Package Pin Locations
								During Reset[6]	After Reset[7]	208 BGA
PJ6	PJ[6] eMIOS[09] PCS_D[5]	134	00 01 10 11	Port J GPIO eMIOS Channel DSPI_D Peripheral Chip Select —	I/O I/O O —	V_{DDE4}	SH	—	—	R10
PJ7	PJ[7] eMIOS[08] PCS_D[1]	135	00 01 10 11	Port J GPIO eMIOS Channel DSPI_D Peripheral Chip Select —	I/O I/O O —	V_{DDE4}	SH	—	—	T10
PJ8	PJ[8] eMIOS[07]	136	00 01 10 11	Port J GPIO eMIOS Channel — —	I/O I/O — —	V_{DDE4}	SH	—	—	T11
PJ9	PJ[9] eMIOS[06]	137	00 01 10 11	Port J GPIO eMIOS Channel — —	I/O I/O — —	V_{DDE4}	SH	—	—	R11
PJ10	PJ[10] eMIOS[05]	138	00 01 10 11	Port J GPIO eMIOS Channel — —	I/O I/O — —	V_{DDE4}	SH	—	—	N12
PJ11	PJ[11] eMIOS[04]	139	00 01 10 11	Port J GPIO eMIOS Channel — —	I/O I/O — —	V_{DDE4}	SH	—	—	P12
PJ12	PJ[12] eMIOS[03]	140	00 01 10 11	Port J GPIO eMIOS Channel — —	I/O I/O — —	V_{DDE4}	SH	—	—	R12
PJ13	PJ[13] eMIOS[02]	141	00 01 10 11	Port J GPIO eMIOS Channel — —	I/O I/O — —	V_{DDE4}	SH	—	—	T12
PJ14	PJ[14] eMIOS[01]	142	00 01 10 11	Port J GPIO eMIOS Channel — —	I/O I/O — —	V_{DDE4}	SH	—	—	R13
PJ15	PJ[15] eMIOS[00]	143	00 01 10 11	Port J GPIO eMIOS Channel — —	I/O I/O — —	V_{DDE4}	SH	—	—	T13

Table 3-1. PXN20 Signal Properties (continued)

Pin Name[1]	Supported Functions[2]	GPIO (PCR) Num[3]	PA[4]	Description	I/O Type	Volt-age	Pad Type[5]	Status		Package Pin Locations
								During Reset[6]	After Reset[7]	208 BGA
Port K (11)										
PK0	PK[0] MLBCLK SCK_B CLKOUT	144	00 01 10 11	Port K GPIO Media Local Bus Clock DSPI_B Serial Clock CLKOUT (Test Only)	I/O I I/O O	V_{DDEMLB}	F	—	—	L1
PK1	PK[1] MLBSIG SOUT_B PCS_D[4]	145	00 01 10 11	Port K GPIO Media Local Bus Signal DSPI_B Serial Data Out DSPI_D Peripheral Chip Select	I/O I/O O O	V_{DDEMLB}	F	—	—	K1
PK2	PK[2] MLBDAT SIN_B PCS_D[5]	146	00 01 10 11	Port K GPIO Media Local Bus Data DSPI_B Serial Data In DSPI_D Peripheral Chip Select	I/O I/O I O	V_{DDEMLB}	F	—	—	K2
PK3	PK[3] FR_A_RX MA[0] PCS_C[1]	147	00 01 10 11	Port K GPIO FlexRay A Receive Data ADC Ext. Mux Address Select DSPI_C Peripheral Chip Select	I/O I O O	V_{DDE2}	SH	—	—	T3
PK4	PK[4] FR_A_TX MA[1] PCS_C[2]	148	00 01 10 11	Port K GPIO FlexRay A Transmit Data ADC Ext. Mux Address Select DSPI_C Peripheral Chip Select	I/O O O O	V_{DDE2}	MH	—	—	R4
PK5	PK[5] FR_A_TX_EN MA[2] PCS_C[3]	149	00 01 10 11	Port K GPIO FlexRay A Transmit Enable ADC Ext. Mux Address Select DSPI_C Peripheral Chip Select	I/O O O O	V_{DDE2}	MH	—	—	T4
PK6	PK[6] FR_B_RX PCS_B[1] PCS_C[4]	150	00 01 10 11	Port K GPIO FlexRay B Receive Data DSPI_B Peripheral Chip Select DSPI_C Peripheral Chip Select	I/O I O O	V_{DDE2}	SH	—	—	R5
PK7	PK[7] FR_B_TX PCS_B[2] PCS_C[5]	151	00 01 10 11	Port K GPIO FlexRay B Transmit Data DSPI_B Peripheral Chip Select DSPI_C Peripheral Chip Select	I/O O O O	V_{DDE2}	MH	—	—	T5
PK8	PK[8] FR_B_TX_EN PCS_B[3] PCS_A[1]	152	00 01 10 11	Port K GPIO FlexRay B Transmit Enable DSPI_B Peripheral Chip Select DSPI_A Peripheral Chip Select	I/O O O O	V_{DDE2}	MH	—	—	R6
PK9	PK[9] CLKOUT PCS_D[1] PCS_A[2] BOOTCFG	153	00 01 10 11	Port K GPIO CLKOUT (User mode) DSPI_D Peripheral Chip Select DSPI_A Peripheral Chip Select Boot Configuration	I/O O O O I	V_{DDE2}	MH	BOOT CFG (Pull-down)	GPIO	T6

Table 3-1. PXN20 Signal Properties (continued)

Pin Name[1]	Supported Functions[2]	GPIO (PCR) Num[3]	PA[4]	Description	I/O Type	Volt-age	Pad Type[5]	Status		Package Pin Locations
								During Reset[6]	After Reset[7]	208 BGA
PK10	PK[10] PCS_B[5] PCS_D[2] PCS_A[3]	154	00 01 10 11	Port K GPIO DSPI_B Peripheral Chip Select DSPI_D Peripheral Chip Select DSPI_A Peripheral Chip Select	I/O O O O	V_{DDE2}	SH	—	—	P6
Miscellaneous Pins (9)										
EXTAL	EXTAL EXTCLK	—	—	Main Crystal Oscillator Input External Clock Input	I I	V_{DDSYN}	A	EXTAL		A14
XTAL	XTAL	—	—	Main Crystal Oscillator Output	O	V_{DDSYN}	A	XTAL		A13
TDI	TDI	—	—	JTAG Test Data Input	I	V_{DDE2}	SH	TDI (Pull Up)		J3
TDO	TDO	—	—	JTAG Test Data Output	O	V_{DDE2}	MH	TDO (Pull Up[8])		M3
TMS	TMS	—	—	JTAG Test Mode Select Input	I	V_{DDE2}	MH	TMS (Pull Up)		L3
TCK	TCK	—	—	JTAG Test Clock Input	I	V_{DDE2}	SH	TCK (Pull Down)		P3
JCOMP	JCOMP	—	—	JTAG Compliancy	I	V_{DDE2}	SH	JCOMP (Pull Down)		K3
TEST	TEST	—	—	Test Mode Select	I	V_{DDE3}	IH	TEST[9]		M13
$\overline{\text{RESET}}$	$\overline{\text{RESET}}$	—	—	External Reset	I/O	V_{DDE1}	MH	$\overline{\text{RESET}}$ (Pull Up)		A11

1 The primary signal name is used as the pin label on the BGA map for identification purposes.

2 Each line in the Signal Name column corresponds to a separate signal function on the pin. For all device I/O pins, the primary, alternate, or GPIO signal functions are designated in the PA field of the System Integration Unit (SIU) PCR registers except where explicitly noted.

3 The GPIO number is the same as the corresponding pad configuration register (SIU_PCR*n*) number.

4 The PA bitfield in the SIU_PCR*n* register selects the signal function for the pin. A dash in the Description field of this table indicates that this value for PC is reserved on this pin, and should not be used.

5 The pad type is indicated by one or more of the following abbreviations: A–analog, F—fast speed, H–high voltage, I—input-only, M–medium speed, S–slow speed. For example, pad type SH designates a slow high-voltage pad.

6 The Status During Reset pin is sampled after the internal POR is negated. Prior to exiting POR, the signal has a high impedance. The terminology used in this column is: O – output, I – input, Up – weak pull up enabled, Down – weak pulldown enabled, Low – output driven low, High – output driven high. A dash on the left side of the slash denotes that both the input and output buffers for the pin are off. A dash on the right side of the slash denotes that there is no weak pull up/down enabled on the pin. The signal name to the left or right of the slash indicates the pin is enabled.

7 The Function After Reset of a GPI function is general purpose input. A dash on the left side of the slash denotes that both the input and output buffers for the pin are off. A dash on the right side of the slash denotes that there is no weak pull up/down enabled on the pin.

8 Pullup is enabled only when JCOMP is negated.

9 Tie to V_{SS} for normal operation.

3.2.1 I/O Power and Ground Segmentation

Table 3-2 gives the preliminary power/ground segmentation. Each segment provides the power and ground for the I/O pins and can be powered by any voltage within the allowed voltage range regardless of the power on the other segments. The power/ground segmentation applies regardless of whether a particular pin is configured for its primary function or GPIO.

Table 3-2. PXN20 Power Segmentation

Pin Name	Function Description	Voltage[1]	Package Pin Locations	
			208	256
V_{DD}	Internal Logic Power	1.2 V	D4, D10, H4, G13, K13, N5	D4, D10, H4, G13, K13, N5
V_{DDE1}	External I/O Power	3.3 or 5.0 V	D6	D6
V_{DDE2}			L4	L4
V_{DDE3}			J13	J13
V_{DDE4}			N10	N10
V_{DDA}	Analog Power	3.3 or 5.0 V	B15	B15
V_{DD33}	3.3 V I/O Power	3.3 V	L13	L13
V_{DDEMLB}	Media Local Bus Power	2.5 or 3.3 V	K4	K4
V_{DDENEX}[2]	Nexus Power	3.3 V	—	E6, K11, L7
V_{RCSEL}	Voltage Regulator Select	V_{SSA} / V_{DDA}	H13	H13
V_{RC}	Voltage Regulator Control Voltage	3.3 or 5.0 V	B10	B10
V_{RCCTL}	Voltage Regulator Control Output	—[3]	B11	B11
V_{DDSYN}	Clock Synthesizer Power	3.3 V	A12	A12
V_{RH}	Analog High Voltage Reference	5.0 V	B16	B16
V_{RL}	Analog Low Voltage Reference	0 V	C16	C16
V_{SS}	Ground	0 V	A1, A16, D7, G4, G[7:10], H[7:10], J[7:10], K[7:10], N13, T1, T16	A1, A16, D7, E[7:12], F[7:12], G4, G[6:12], H[7:12], J[7:12], K[6:10], K12, L[8:10], L12, N13, T1, T16
V_{SSA}	Analog Ground	0 V	C15	C15
V_{SSSYN}	Clock Synthesizer Ground	0 V	A15	A15

1 Nominal voltages.

2 Dedicated Nexus power pin on 256-pin package only. On the 208-pin package, VDDENEX is tied to VSS internal to the package substrate and is not available externally.

3 Base current to external NPN power transistor. Voltage may vary.

3.3 Pinout

Figure 3-2 shows the 208-ball MAPBGA pin assignments. Figure 3-3 shows the 256-ball MAPBGA pin assignments. For more information, see the *PXN20 Microcontroller Data Sheet*.

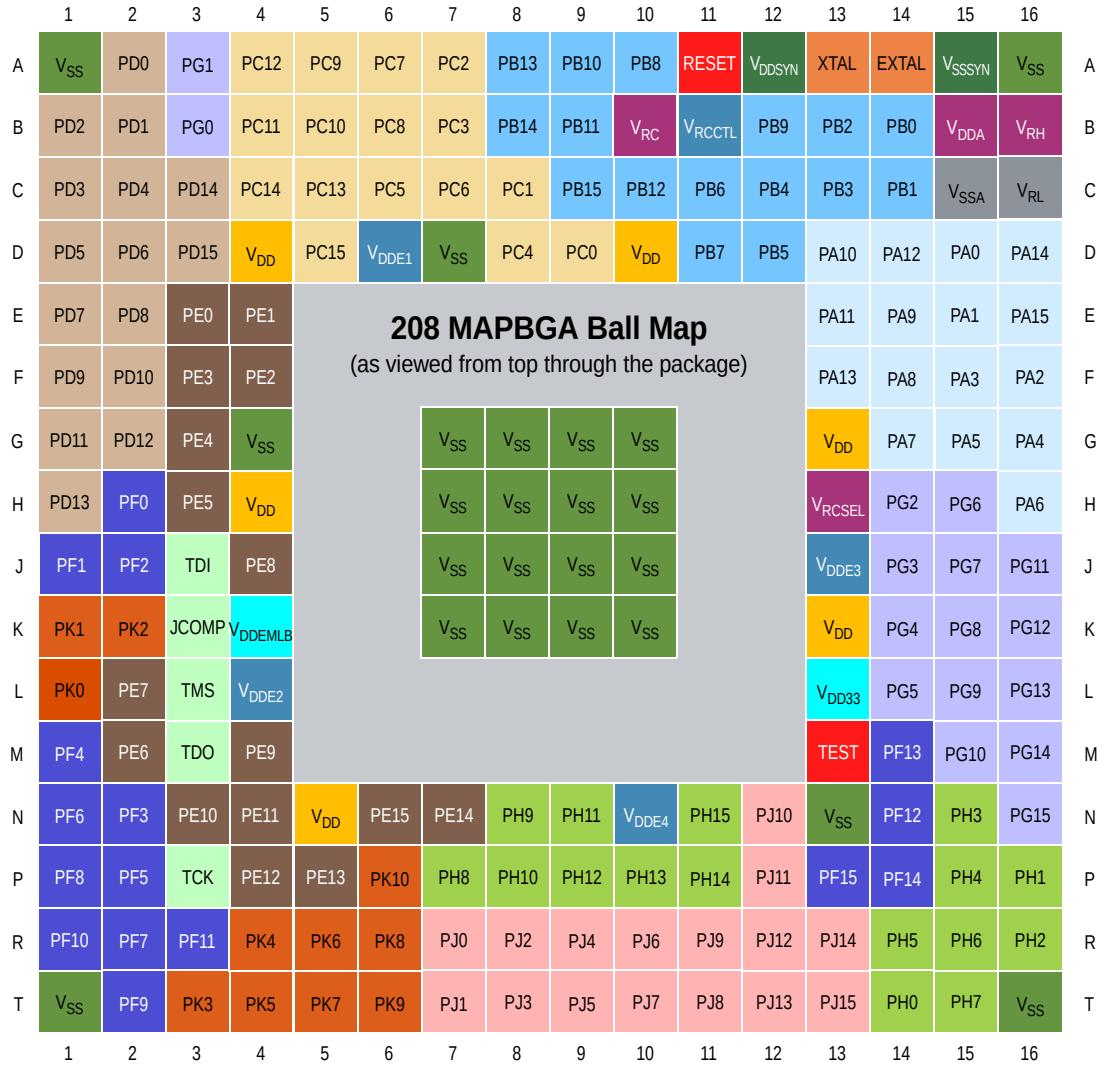


Figure 3-2. PXN20 Pinout – 208 MAPBGA

256 MAPBGA Ball Map

(as viewed from top through the package)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	
A	V_{SS}	PD0	PG1	PC12	PC9	PC7	PC2	PB13	PB10	PB8	RESET	V_{DDSYN}	XTAL	EXTAL	V_{SSSYN}	V_{SS}	A
B	PD2	PD1	PG0	PC11	PC10	PC8	PC3	PB14	PB11	V_{RC}	V_{RCCTL}	PB9	PB2	PB0	V_{DDA}	V_{RH}	B
C	PD3	PD4	PD14	PC14	PC13	PC5	PC6	PC1	PB15	PB12	PB6	PB4	PB3	PB1	V_{SSA}	V_{RL}	C
D	PD5	PD6	PD15	V_{DD}	PC15	V_{DDE1}	V_{SS}	PC4	PC0	V_{DD}	PB7	PB5	PA10	PA12	PA0	PA14	D
E	PD7	PD8	PE0	PE1	MDO0	V_{DDENEX}	V_{SS}	V_{SS}	V_{SS}	V_{SS}	V_{SS}	V_{SS}	PA11	PA9	PA1	PA15	E
F	PD9	PD10	PE3	PE2	MDO1	V_{SS}	V_{SS}	V_{SS}	V_{SS}	V_{SS}	V_{SS}	V_{SS}	PA13	PA8	PA3	PA2	F
G	PD11	PD12	PE4	V_{SS}	MDO2	V_{SS}	V_{SS}	V_{SS}	V_{SS}	V_{SS}	V_{SS}	V_{SS}	V_{DD}	PA7	PA5	PA4	G
H	PD13	PF0	PE5	V_{DD}	MDO3	MDO4	V_{SS}	V_{SS}	V_{SS}	V_{SS}	V_{SS}	V_{SS}	V_{RCSEL}	PG2	PG6	PA6	H
J	PF1	PF2	TDI	PE8	MDO6	MDO5	V_{SS}	V_{SS}	V_{SS}	V_{SS}	V_{SS}	V_{SS}	V_{DDE3}	PG3	PG7	PG11	J
K	PK1	PK2	JCOMP	V_{DDEMLB}	MDO7	V_{SS}	V_{SS}	V_{SS}	V_{SS}	V_{SS}	V_{DDENEX}	V_{SS}	V_{DD}	PG4	PG8	PG12	K
L	PK0	PE7	TMS	V_{DDE2}	MDO8	V_{SS}	V_{DDENEX}	V_{SS}	V_{SS}	V_{SS}	V_{SS}	V_{SS}	V_{DD33}	PG5	PG9	PG13	L
M	PF4	PE6	TDO	PE9	MDO9	MDO10	MDO11	MSEO1	MSEO0	MCKO	EVTI	EVTO	TEST	PF13	PG10	PG14	M
N	PF6	PF3	PE10	PE11	V_{DD}	PE15	PE14	PH9	PH11	V_{DDE4}	PH15	PJ10	V_{SS}	PF12	PH3	PG15	N
P	PF8	PF5	TCK	PE12	PE13	PK10	PH8	PH10	PH12	PH13	PH14	PJ11	PF15	PF14	PH4	PH1	P
R	PF10	PF7	PF11	PK4	PK6	PK8	PJ0	PJ2	PJ4	PJ6	PJ9	PJ12	PJ14	PH5	PH6	PH2	R
T	V_{SS}	PF9	PK3	PK5	PK7	PK9	PJ1	PJ3	PJ5	PJ7	PJ8	PJ13	PJ15	PH0	PH7	V_{SS}	T
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	

Figure 3-3. PXN20 Pinout – 256-pin MAPBGA (emulation device only)

3.4 Detailed Signal Description

This section provides detailed descriptions of the signal functions available for the device.

3.4.1 Port A Pins

3.4.1.1 PA0 to PA13 — GPI (PA[0:13]) / Analog Input (AN[0:13])

PA[0:13] are general-purpose input (GPI) pins. AN[0:13] are single-ended analog input pins.

3.4.1.2 PA14 — GPI (PA[14]) / Analog Input (AN[14]) / 32 kHz Crystal Input (EXTAL32)

PA[14] is a general-purpose input (GPI) pin. AN[14] is a single-ended analog input pin. EXTAL32 is the input pin for an external 32 kHz crystal oscillator.

3.4.1.3 PA15 — GPI (PA[15]) / Analog Input (AN[15]) / 32 kHz Crystal Output (XTAL32)

PA[15] is a GPI pin. AN[15] is a single-ended analog input pin. XTAL32 is the output pin for an external 32 kHz crystal oscillator.

3.4.2 Port B Pins

3.4.2.1 PB0 — GPIO (PB[0]) / Analog Input (AN[16]) / Analog Input Channel for External Mux (ANW)

PB[0] is a GPIO pin. AN[16] is a single-ended analog input pin. ANW is an input channel for the ADC external multiplexer.

3.4.2.2 PB1 — GPIO (PB[1]) / Analog Input (AN[17]) / Analog Input Channel for External Mux (ANX)

PB[1] is a GPIO pin. AN[17] is a single-ended analog input pin. ANX is an input channel for the ADC external multiplexer.

3.4.2.3 PB2 — GPIO (PB[2]) / Analog Input (AN[18]) / Analog Input Channel for External Mux (ANY)

PB[2] is a GPIO pin. AN[18] is a single-ended analog input pin. ANY is an input channel for the ADC external multiplexer.

3.4.2.4 PB3 — GPIO (PB[3]) / Analog Input (AN[19]) / Analog Input Channel for External Mux (ANZ)

PB[3] is a GPIO pin. AN[19] is a single-ended analog input pin. ANZ is an input channel for the ADC external multiplexer.

3.4.2.5 PB4 to PB7 — GPIO (PB[4:7]) / Analog Input (AN[20:23])

PB[4:7] are GPIO pins. AN[20:23] are single-ended analog input pins. PB[4:7] can be configured as wakeup pins in the CRP_PWKENL register.

3.4.2.6 PB8 — GPIO (PB[8]) / Analog Input (AN[24]) / DSPI_A Peripheral Chip Select (PCS_A[2])

PB[8] is a GPIO pin. AN[24] is a single-ended analog input pin. PCS_A[2] is a peripheral chip select output pin for the DSPI A module.

3.4.2.7 PB9 — GPIO (PB[9]) / Analog Input (AN[25]) / DSPI_A Peripheral Chip Select (PCS_A[3])

PB[9] is a GPIO pin. AN[25] is a single-ended analog input pin. PCS_A[3] is a peripheral chip select output pin for the DSPI A module.

3.4.2.8 PB10 — GPIO (PB[10]) / Analog Input (AN[26]) / DSPI_B Peripheral Chip Select (PCS_B[4])

PB[10] is a GPIO pin. AN[26] is a single-ended analog input pin. PCS_B[4] is a peripheral chip select output pin for the DSPI B module.

3.4.2.9 PB11 — GPIO (PB[11]) / Analog Input (AN[27]) / DSPI_B Peripheral Chip Select (PCS_B[5])

PB[11] is a GPIO pin. AN[27] is a single-ended analog input pin. PCS_B[5] is a peripheral chip select output pin for the DSPI B module.

3.4.2.10 PB12 — GPIO (PB[12]) / Analog Input (AN[28]) / DSPI_C Peripheral Chip Select (PCS_C[1])

PB[12] is a GPIO pin. AN[28] is a single-ended analog input pin. PCS_C[1] is a peripheral chip select output pin for the DSPI C module.

3.4.2.11 PB13 — GPIO (PB[13]) / Analog Input (AN[29]) / DSPI_C Peripheral Chip Select (PCS_C[2])

PB[13] is a GPIO pin. AN[29] is a single-ended analog input pin. PCS_C[2] is a peripheral chip select output pin for the DSPI C module.

3.4.2.12 PB14 — GPIO (PB[14]) / Analog Input (AN[30]) / DSPI_D Peripheral Chip Select (PCS_D[3])

PB[14] is a GPIO pin. AN[30] is a single-ended analog input pin. PCS_D[3] is a peripheral chip select output pin for the DSPI D module.

3.4.2.13 PB15 — GPIO (PB[15]) / Analog Input (AN[31]) / DSPI_D Peripheral Chip Select (PCS_D[4])

PB[15] is a GPIO pin. AN[31] is a single-ended analog input pin. PCS_D[4] is a peripheral chip select output pin for the DSPI D module.

3.4.3 Port C Pins

3.4.3.1 PC0 to PC1 — GPIO (PC[0:1]) / Analog Input (AN[32:33])

PC[0:1] are GPIO pins. AN[32:33] are single-ended analog input pins.

3.4.3.2 PC2 — GPIO (PC[2]) / Analog Input (AN[34]) / Nexus Event In ($\overline{\text{EVTI}}$)

PC[2] is a GPIO pin. AN[34] is a single-ended analog input pin. $\overline{\text{EVTI}}$ is the Nexus Event Input pin.

3.4.3.3 PC3 — GPIO (PC[3]) / Analog Input (AN[35]) / Nexus Event Out ($\overline{\text{EVTO}}$)

PC[3] is a GPIO pin. AN[35] is a single-ended analog input pin. $\overline{\text{EVTO}}$ is the Nexus Event Output pin.

3.4.3.4 PC4 — GPIO (PC[4]) / Analog Input (AN[36])

PC[4] is a GPIO pin. AN[36] is a single-ended analog input pin.

3.4.3.5 PC5 — GPIO (PC[5]) / Analog Input (AN[37]) / Z6 Non-Maskable Interrupt (Z6_NMI)

PC[5] is a GPIO pin. AN[37] is a single-ended analog input pin. Z6_NMI is the non-maskable interrupt input pin for the Z6 core.

3.4.3.6 PC6 — GPIO (PC[6]) / Analog Input (AN[38]) / Z0 Non-Maskable Interrupt (Z0_NMI)

PC[6] is a GPIO pin. AN[38] is a single-ended analog input pin. Z0_NMI is the non-maskable interrupt input pin for the Z0 core.

3.4.3.7 PC7 — GPIO (PC[7]) / Analog Input (AN[39]) / FlexRay Debug 3 (FR_DBG[3])

PC[7] is a GPIO pin. AN[39] is a single-ended analog input pin. FR_DBG[3] is one of the FlexRay debug output port pins.

3.4.3.8 PC8 — GPIO (PC[8]) / Analog Input (AN[40]) / FlexRay Debug 2 (FR_DBG[2])

PC[8] is a GPIO pin. AN[40] is a single-ended analog input pin. FR_DBG[2] is one of the FlexRay debug output port pins.

3.4.3.9 PC9 — GPIO (PC[9]) / Analog Input (AN[41]) / FlexRay Debug 1 (FR_DBG[1])

PC[9] is a GPIO pin. AN[41] is a single-ended analog input pin. FR_DBG[1] is one of the FlexRay debug output port pins.

3.4.3.10 PC10 — GPIO (PC[10]) / Analog Input (AN[42]) / FlexRay Debug 0 (FR_DBG[0])

PC[10] is a GPIO pin. AN[42] is a single-ended analog input pin. FR_DBG[0] is one of the FlexRay debug output port pins.

3.4.3.11 PC11 — GPIO (PC[11]) / Analog Input (AN[43]) / I^2C_C Serial Clock Line (SCL_C)

PC[11] is a GPIO pin. AN[43] is a single-ended analog input pin. SCL_C is the serial clock signal for the I^2C_C module.

3.4.3.12 PC12 — GPIO (PC[12]) / Analog Input (AN[44]) / I^2C_C Serial Data Line (SDA_C)

PC[12] is a GPIO pin. AN[44] is a single-ended analog input pin. SDA_C is the serial data line for the I^2C_B module.

3.4.3.13 PC13 — GPIO (PC[13]) / Analog Input (AN[45]) / External Analog Mux Address Output (MA[0])

PC[13] is a GPIO pin. AN[45] is a single-ended analog input pin. MA[0] is an address output for an external analog multiplexer used to select the multiplexer input channel to connect to the ADC.

3.4.3.14 PC14 — GPIO (PC[14]) / Analog Input (AN[46]) / External Analog Mux Address Output (MA[1])

PC[14] is a GPIO pin. AN[46] is a single-ended analog input pin. MA[1] is an address output for an external analog multiplexer used to select the multiplexer input channel to connect to the ADC.

3.4.3.15 PC15 — GPIO (PC[15]) / Analog Input (AN[47]) / External Analog Mux Address Output (MA[2])

PC[15] is a GPIO pin. AN[47] is a single-ended analog input pin. MA[2] is an address output for an external analog multiplexer used to select the multiplexer input channel to connect to the ADC.

3.4.4 Port D Pins

3.4.4.1 PD0 — GPIO (PD[0]) / CAN_A Transmit (CNTX_A)

PD[0] is a GPIO pin. CNTX_A is the transmit output pin for the FlexCAN A module.

3.4.4.2 PD1 — GPIO (PD[1]) / CAN_A Receive (CNRX_A)

PD[1] is a GPIO pin. CNRX_A is the receive input pin for the FlexCAN A module. PD[1] can be configured as a wakeup pin in the CRP_PWKENL register.

3.4.4.3 PD2 — GPIO (PD[2]) / CAN_B Transmit (CNTX_B)

PD[2] is a GPIO pin. CNTX_B is the transmit output pin for the FlexCAN B module.

3.4.4.4 PD3 — GPIO (PD[3]) / CAN_B Receive (CNRX_B)

PD[3] is a GPIO pin. CNRX_B is the receive input pin for the FlexCAN B module. PD[3] can be configured as a wakeup pin in the CRP_PWKENL register.

3.4.4.5 PD4 — GPIO (PD[4]) / CAN_C Transmit (CNTX_C)

PD[4] is a GPIO pin. CNTX_C is the transmit output pin for the FlexCAN C module.

3.4.4.6 PD5 — GPIO (PD[5]) / CAN_C Receive (CNRX_C)

PD[5] is a GPIO pin. CNRX_C is the receive input pin for the FlexCAN C module. PD[5] can be configured as a wakeup pin in the CRP_PWKENL register.

3.4.4.7 PD6 — GPIO (PD[6]) / CAN_D Transmit (CNTX_D) / TXD_K / I^2C_B Serial Clock Line (SCL_B)

PD[6] is a GPIO pin. CNTX_D is the transmit output pin for the FlexCAN D module. TXD_K is the transmit output pin for the eSCI K module. SCL_B is the serial clock signal for the I^2C B module.

3.4.4.8 PD7 — GPIO (PD[7]) / CAN_D Receive (CNRX_D) / RXD_K / I^2C_B Serial Data Line (SDA_B)

PD[7] is a GPIO pin. CNRX_D is the receive input pin for the FlexCAN D module. RXD_K is the receive input pin for the eSCI K module. SDA_B is the serial data line for the I^2C B module. PD[7] can be configured as a wakeup pin in the CRP_PWKENL register.

3.4.4.9 PD8 — GPIO (PD[8]) / CAN_E Transmit (CNTX_E) / TXD_LK / I^2C_C Serial Clock Line (SCL_C)

PD[8] is a GPIO pin. CNTX_E is the transmit output pin for the FlexCAN E module. TXD_L is the transmit output pin for the eSCI L module. SCL_C is the serial clock signal for the I^2C C module.

3.4.4.10 PD9 — GPIO (PD[9]) / CAN_E Receive (CNRX_E) / RXD_L / I2C_C Serial Data Line (SDA_C)

PD[9] is a GPIO pin. CNRX_E is the receive input pin for the FlexCAN E module. RXD_L is the receive input pin for the eSCI L module.SDA_C is the serial data line for the I2C C module. PD[9] can be configured as a wakeup pin in the CRP_PWKENL register.

3.4.4.11 PD10 — GPIO (PD[10]) / CAN_F Transmit (CNTX_F) / TXD_M / I2C_D Serial Clock Line (SCL_D)

PD[10] is a GPIO pin. CNTX_F is the transmit output pin for the FlexCAN F module. TXD_M is the transmit output pin for the eSCI M module. SCL_D is the serial clock signal for the I2C D module.

3.4.4.12 PD11 — GPIO (PD[11]) / CAN_F Receive (CNRX_F) / RXD_M / I2C_D Serial Data Line (SDA_D)

PD[11] is a GPIO pin. CNRX_F is the receive input pin for the FlexCAN F module. RXD_M is the receive input pin for the eSCI M module.SDA_D is the serial data line for the I2C D module. PD[11] can be configured as a wakeup pin in the CRP_PWKENL register.

3.4.4.13 PD12 — GPIO (PD[12]) / eSCI_A Transmit (TXD_A)

PD[12] is a GPIO pin. TXD_A is the transmit output pin for the eSCI A module.

3.4.4.14 PD13 — GPIO (PD[13]) / eSCI_A Receive (RXD_A)

PD[13] is a GPIO pin. RXD_A is the receive input pin for the eSCI A module. PD[13] can be configured as a wakeup pin in the CRP_PWKENL register.

3.4.4.15 PD14 — GPIO (PD[14]) / eSCI_B Transmit (TXD_B)

PD[14] is a GPIO pin. TXD_B is the transmit output pin for the eSCI B module.

3.4.4.16 PD15 — GPIO (PD[15]) / eSCI_B Receive (RXD_B)

PD[15] is a GPIO pin. RXD_B is the receive input pin for the eSCI B module. PD[15] can be configured as a wakeup pin in the CRP_PWKENL register.

3.4.5 Port E Pins

3.4.5.1 PE0 — GPIO (PE[0]) / eSCI_C Transmit (TXD_C) / eMIOS Channel (eMIOS[31])

PE[0] is a GPIO pin. TXD_C is the transmit output pin for the eSCI C module. eMIOS[31] is an input/output channel pin for the eMIOS200 module.

3.4.5.2 PE1 — GPIO (PE[1]) / eSCI_C Receive (RXD_C) / eMIOS Channel (eMIOS[30])

PE[1] is a GPIO pin. RXD_C is the receive input pin for the eSCI C module. eMIOS[30] is an input/output channel pin for the eMIOS200 module. PE[1] can be configured as a wakeup pin in the CRP_PWKENL register.

3.4.5.3 PE2 — GPIO (PE[2]) / eSCI_D Transmit (TXD_D) / eMIOS Channel (eMIOS[29])

PE[2] is a GPIO pin. TXD_D is the transmit output pin for the eSCI D module. eMIOS[29] is an input/output channel pin for the eMIOS200 module.

3.4.5.4 PE3 — GPIO (PE[3]) / eSCI_D Receive (RXD_D) / eMIOS Channel (eMIOS[28])

PE[3] is a GPIO pin. RXD_D is the receive input pin for the eSCI D module. eMIOS[28] is an input/output channel pin for the eMIOS200 module. PE[3] can be configured as a wakeup pin in the CRP_PWKENL register.

3.4.5.5 PE4 — GPIO (PE[4]) / eSCI_E Transmit (TXD_E) / eMIOS Channel (eMIOS[27])

PE[4] is a GPIO pin. TXD_E is the transmit output pin for the eSCI E module. eMIOS[27] is an input/output channel pin for the eMIOS200 module.

3.4.5.6 PE5 — GPIO (PE[5]) / eSCI_E Receive (RXD_E) / eMIOS Channel (eMIOS[26])

PE[5] is a GPIO pin. RXD_E is the receive input pin for the eSCI E module. eMIOS[26] is an input/output channel pin for the eMIOS200 module. PE[5] can be configured as a wakeup pin in the CRP_PWKENL register.

3.4.5.7 PE6 — GPIO (PE[6]) / eSCI_F Transmit (TXD_F) / eMIOS Channel (eMIOS[25])

PE[6] is a GPIO pin. TXD_F is the transmit output pin for the eSCI F module. eMIOS[25] is an input/output channel pin for the eMIOS200 module.

3.4.5.8 PE7 — GPIO (PE[7]) / eSCI_F Receive (RXD_F) / eMIOS Channel (eMIOS[24])

PE[7] is a GPIO pin. RXD_F is the receive input pin for the eSCI F module. eMIOS[24] is an input/output channel pin for the eMIOS200 module. PE[7] can be configured as a wakeup pin in the CRP_PWKENL register.

3.4.5.9 PE8 — GPIO (PE[8]) / eSCI_G Transmit (TXD_G) / DSPI_A Peripheral Chip Select (PCS_A[1])

PE[8] is a GPIO pin. TXD_G is the transmit output pin for the eSCI G module. PCS_A[1] is a peripheral chip select output pin for the DSPI A module.

3.4.5.10 PE9 — GPIO (PE[9]) / eSCI_G Receive (RXD_G) / DSPI_A Peripheral Chip Select (PCS_A[4])

PE[9] is a GPIO pin. RXD_G is the receive input pin for the eSCI G module. PCS_A[4] is a peripheral chip select output pin for the DSPI A module. PE[9] can be configured as a wakeup pin in the CRP_PWKENH register.

3.4.5.11 PE10 — GPIO (PE[10]) / eSCI_H Transmit (TXD_H) / DSPI_B Peripheral Chip Select (PCS_B[3])

PE[10] is a GPIO pin. TXD_H is the transmit output pin for the eSCI H module. PCS_B[3] is a peripheral chip select output pin for the DSPI B module.

3.4.5.12 PE11 — GPIO (PE[11]) / eSCI_H Receive (RXD_H) / DSPI_B Peripheral Chip Select (PCS_B[2])

PE[11] is a GPIO pin. RXD_H is the receive input pin for the eSCI H module. PCS_B[2] is a peripheral chip select output pin for the DSPI B module. PE[11] can be configured as a wakeup pin in the CRP_PWKENH register.

3.4.5.13 PE12 — GPIO (PE[12]) / eSCI_J Transmit (TXD_J) / DSPI_C Peripheral Chip Select (PCS_C[5])

PE[12] is a GPIO pin. TXD_J is the transmit output pin for the eSCI J module. PCS_C[5] is a peripheral chip select output pin for the DSPI C module.

3.4.5.14 PE13 — GPIO (PE[13]) / eSCI_J Receive (RXD_J) / DSPI_C Peripheral Chip Select (PCS_C[3])

PE[13] is a GPIO pin. RXD_J is the receive input pin for the eSCI J module. PCS_C[3] is a peripheral chip select output pin for the DSPI C module. PE[13] can be configured as a wakeup pin in the CRP_PWKENH register.

3.4.5.15 PE14 — GPIO (PE[14]) / I^2C_A Serial Clock Line (SCL_A) / DSPI_D Peripheral Chip Select (PCS_D[2])

PE[14] is a GPIO pin. SCL_A is the serial clock signal for the I^2C A module. PCS_D[2] is a peripheral chip select output pin for the DSPI D module.

3.4.5.16 PE15 — GPIO (PE[15]) / I2C_A Serial Data Line (SDA_A) / DSPI_D Peripheral Chip Select (PCS_D[5])

PE[15] is a GPIO pin. SDA_A is the serial data line for the I2C A module. PCS_D[5] is a peripheral chip select output pin for the DSPI D module.

3.4.6 Port F Pins

3.4.6.1 PF0 — GPIO (PF[0]) / DSPI_A Clock (SCK_A)

PF[0] is a GPIO pin. SCK_A is the SPI clock pin for the DSPI A module.

3.4.6.2 PF1 — GPIO (PF[1]) / DSPI_A Data Output (SOUT_A)

PF[1] is a GPIO pin. SOUT_A is the data output pin for the DSPI A module.

3.4.6.3 PF2 — GPIO (PF[2]) / DSPI_A Data Input (SIN_A)

PF[2] is a GPIO pin. SIN_A is the data input pin for the DSPI A module.

3.4.6.4 PF3 — GPIO (PF[3]) / DSPI_A Peripheral Chip Select (PCS_A[0]) / DSPI_B Peripheral Chip Select (PCS_B[5]) / DSPI_C Peripheral Chip Select (PCS_C[4])

PF[3] is a GPIO pin. PCS_A[0] is a peripheral chip select input/output pin for the DSPI A module. PCS_B[5] is a peripheral chip select output pin for the DSPI B module. PCS_C[4] is a peripheral chip select output pin for the DSPI C module. PF[3] can be configured as a wakeup pin in the CRP_PWKENH register.

3.4.6.5 PF4 — GPIO (PF[4]) / DSPI_B Clock (SCK_B) / DSPI_A Peripheral Chip Select (PCS_A[1]) / DSPI_C Peripheral Chip Select (PCS_C[2])

PF[4] is a GPIO pin. SCK_B is the SPI clock pin for the DSPI B module. PCS_A[1] is a peripheral chip select output pin for the DSPI A module. PCS_C[2] is a peripheral chip select output pin for the DSPI C module.

3.4.6.6 PF5 — GPIO (PF[5]) / DSPI_B Data Output (SOUT_B) / DSPI_A Peripheral Chip Select (PCS_A[2]) / DSPI_C Peripheral Chip Select (PCS_C[3])

PF[5] is a GPIO pin. SOUT_B is the data output pin for the DSPI B module. PCS_A[2] is a peripheral chip select output pin for the DSPI A module. PCS_C[3] is a peripheral chip select output pin for the DSPI C module.

3.4.6.7 PF6 — GPIO (PF[6]) / DSPI_B Data Input (SIN_B) / DSPI_A Peripheral Chip Select (PCS_A[3]) / DSPI_C Peripheral Chip Select (PCS_C[5])

PF[6] is a GPIO pin. SIN_B is the data input pin for the DSPI B module. PCS_A[3] is a peripheral chip select output pin for the DSPI A module. PCS_C[5] is a peripheral chip select output pin for the DSPI C module.

3.4.6.8 PF7 — GPIO (PF[7]) / DSPI_B Peripheral Chip Select (PCS_B[0]) / DSPI_C Peripheral Chip Select / (PCS_C[5]) / DSPI_D Peripheral Chip Select (PCS_D[4])

PF[7] is a GPIO pin. PCS_B[0] is a peripheral chip select input/output pin for the DSPI B module. PCS_C[5] is a peripheral chip select output pin for the DSPI C module. PCS_D[4] is a peripheral chip select output pin for the DSPI D module. PF[7] can be configured as a wakeup pin in the CRP_PWKENH register.

3.4.6.9 PF8 — GPIO (PF[8]) / DSPI_C Clock (SCK_C)

PF[8] is a GPIO pin. SCK_C is the SPI clock pin for the DSPI C module.

3.4.6.10 PF9 — GPIO (PF[9]) / DSPI_C Data Output (SOUT_C)

PF[9] is a GPIO pin. SOUT_C is the data output pin for the DSPI C module.

3.4.6.11 PF10 — GPIO (PF[10]) / DSPI_C Data Input (SIN_C)

PF[10] is a GPIO pin. SIN_C is the data input pin for the DSPI C module.

3.4.6.12 PF11 — GPIO (PF[11]) / DSPI_C Peripheral Chip Select (PCS_C[0]) / DSPI_D Peripheral Chip Select / (PCS_D[5]) / DSPI_A Peripheral Chip Select (PCS_A[4])

PF[11] is a GPIO pin. PCS_C[0] is a peripheral chip select input/output pin for the DSPI C module. PCS_D[5] is a peripheral chip select output pin for the DSPI D module. PCS_A[4] is a peripheral chip select output pin for the DSPI A module. PF[11] can be configured as a wakeup pin in the CRP_PWKENH register.

3.4.6.13 PF12 — GPIO (PF[12]) / DSPI_D Clock (SCK_D)

PF[12] is a GPIO pin. SCK_D is the SPI clock pin for the DSPI D module.

3.4.6.14 PF13 — GPIO (PF[13]) / DSPI_D Data Output (SOUT_D)

PF[13] is a GPIO pin. SOUT_D is the data output pin for the DSPI D module.

3.4.6.15 PF14 — GPIO (PF[14]) / DSPI_D Data Input (SIN_D)

PF[14] is a GPIO pin. SIN_D is the data input pin for the DSPI D module.

3.4.6.16 PF15 — GPIO (PF[15]) / DSPI_D Peripheral Chip Select (PCS_D[0]) / DSPI_A Peripheral Chip Select (PCS_A[5]) / DSPI_B Peripheral Chip Select (PCS_B[4])

PF[15] is a GPIO pin. PCS_D[0] is a peripheral chip select input/output pin for the DSPI D module. PCS_A[5] is a peripheral chip select output pin for the DSPI A module. PCS_B[4] is a peripheral chip select output pin for the DSPI B module. PF[15] can be configured as a wakeup pin in the CRP_PWKENH register.

3.4.7 Port G Pins

3.4.7.1 PG0 — GPIO (PG[0]) / DSPI_A Peripheral Chip Select (PCS_A[4]) / DSPI_B Peripheral Chip Select (PCS_B[3]) / Analog Input (AN[48])

PG[0] is a GPIO pin. PCS_A[4] is a peripheral chip select output pin for the DSPI A module. PCS_B[3] is a peripheral chip select output pin for the DSPI B module. AN[48] is a single-ended analog input pin.

3.4.7.2 PG1 — GPIO (PG[1]) / DSPI_A Peripheral Chip Select (PCS_A[5]) / DSPI_B Peripheral Chip Select (PCS_B[4]) / Analog Input (AN[49])

PG[1] is a GPIO pin. PCS_A[5] is a peripheral chip select output pin for the DSPI A module. PCS_B[4] is a peripheral chip select output pin for the DSPI B module. AN[49] is a single-ended analog input pin.

3.4.7.3 PG2 — GPIO (PG[2]) / DSPI_D Peripheral Chip Select (PCS_D[1]) / I^2C_C Serial Clock Line (SCL_C) / Analog Input (AN[50])

PG[2] is a GPIO pin. PCS_D[1] is a peripheral chip select output pin for the DSPI D module. SCL_C is the serial clock signal for the I^2C_C module. AN[50] is a single-ended analog input pin.

3.4.7.4 PG3 — GPIO (PG[3]) / DSPI_D Peripheral Chip Select (PCS_D[2]) / I^2C_C Serial Data Line (SDA_C) / Analog Input (AN[51])

PG[3] is a GPIO pin. PCS_D[2] is a peripheral chip select output pin for the DSPI D module. SDA_C is the serial data line for the I^2C_C module. AN[51] is a single-ended analog input pin.

3.4.7.5 PG4 — GPIO (PG[4]) / DSPI_D Peripheral Chip Select (PCS_D[3]) / I^2C_B Serial Clock Line (SCL_B) / Analog Input (AN[52])

PG[4] is a GPIO pin. PCS_D[3] is a peripheral chip select output pin for the DSPI D module. SCL_B is the serial clock signal for the I^2C_B module. AN[52] is a single-ended analog input pin.

3.4.7.6 PG5 — GPIO (PG[5]) / DSPI_D Peripheral Chip Select (PCS_D[4]) / I2C_B Serial Data Line (SDA_B) / Analog Input (AN[53])

PG[5] is a GPIO pin. PCS_D[4] is a peripheral chip select output pin for the DSPI D module. SDA_B is the serial data line for the I2C_B module. AN[53] is a single-ended analog input pin.

3.4.7.7 PG6 — GPIO (PG[6]) / DSPI_C Peripheral Chip Select (PCS_C[1]) / Ethernet Management Data Clock (FEC_MDC) / Analog Input (AN[54])

PG[6] is a GPIO pin. PCS_C[1] is a peripheral chip select output pin for the DSPI C module. FEC_MDC is the Ethernet management data clock output pin. AN[54] is a single-ended analog input pin.

3.4.7.8 PG7 — GPIO (PG[7]) / DSPI_C Peripheral Chip Select (PCS_C[2]) / Ethernet Management Data I/O (FEC_MDIO) / Analog Input (AN[55])

PG[7] is a GPIO pin. PCS_C[2] is a peripheral chip select output pin for the DSPI C module. FEC_MDIO is the Ethernet management data I/O pin. AN[55] is a single-ended analog input pin.

3.4.7.9 PG8 — GPIO (PG[8]) / eMIOS Channel (eMIOS[7]) / Ethernet Transmit Clock (FEC_TX_CLK) / Analog Input (AN[56])

PG[8] is a GPIO pin. eMIOS[7] is an input/output channel pin for the eMIOS200 module. FEC_TX_CLK is the Ethernet transmit clock input pin. AN[56] is a single-ended analog input pin.

3.4.7.10 PG9 — GPIO (PG[9]) / eMIOS Channel (eMIOS[6]) / Ethernet Carrier Sense (FEC_CRS) / Analog Input (AN[57])

PG[9] is a GPIO pin. eMIOS[6] is an input/output channel pin for the eMIOS200 module. FEC_CRS is the Ethernet carrier sense input pin. AN[57] is a single-ended analog input pin.

3.4.7.11 PG10 — GPIO (PG[10]) / eMIOS Channel (eMIOS[5]) / Ethernet Transmit Error (FEC_TX_ER) / Analog Input (AN[58])

PG[10] is a GPIO pin.eMIOS[5] is an input/output channel pin for the eMIOS200 module. FEC_TX_ER is the Ethernet transmit error output pin. AN[58] is a single-ended analog input pin.

3.4.7.12 PG11 — GPIO (PG[11]) / eMIOS Channel (eMIOS[4]) / Ethernet Receive Clock (FEC_RX_CLK) / Analog Input (AN[59])

PG[11] is a GPIO pin. eMIOS[4] is an input/output channel pin for the eMIOS200 module. FEC_RX_CLK is the Ethernet receive clock input pin. AN[59] is a single-ended analog input pin.

3.4.7.13 PG12 — GPIO (PG[12]) / eMIOS Channel (eMIOS[3]) / Ethernet Transmit Data (FEC_TXD[0]) / Analog Input (AN[60])

PG[12] is a GPIO pin. eMIOS[3] is an input/output channel pin for the eMIOS200 module. FEC_TXD[0] is an Ethernet transmit data output pin. AN[60] is a single-ended analog input pin.

3.4.7.14 PG13 — GPIO (PG[13]) / eMIOS Channel (eMIOS[2]) / Ethernet Transmit Data (FEC_TXD[1]) / Analog Input (AN[61])

PG[13] is a GPIO pin. eMIOS[2] is an input/output channel pin for the eMIOS200 module. FEC_TXD[1] is an Ethernet transmit data output pin. AN[61] is a single-ended analog input pin.

3.4.7.15 PG14 — GPIO (PG[14]) / eMIOS Channel (eMIOS[1] / Ethernet Transmit Data (FEC_TXD[2]) / Analog Input (AN[62])

PG[14] is a GPIO pin. eMIOS[1] is an input/output channel pin for the eMIOS200 module. FEC_TXD[2] is an Ethernet transmit data output pin. AN[62] is a single-ended analog input pin.

3.4.7.16 PG15 — GPIO (PG[15]) / eMIOS Channel (eMIOS[0]) / Ethernet Transmit Data (FEC_TXD[3]) / Analog Input (AN[63])

PG[15] is a GPIO pin. eMIOS[0] is an input/output channel pin for the eMIOS200 module. FEC_TXD[3] is an Ethernet transmit data output pin. AN[63] is a single-ended analog input pin.

3.4.8 Port H Pins

3.4.8.1 PH0 — GPIO (PH[0]) / eMIOS Channel (eMIOS[31]) / Ethernet Collision (FEC_COL)

PH[0] is a GPIO pin. eMIOS[31] is an input/output channel pin for the eMIOS200 module. FEC_COL is the Ethernet collision output pin.

3.4.8.2 PH1 — GPIO (PH[1]) / eMIOS Channel (eMIOS[30]) / Ethernet Receive Data Valid (FEC_RX_DV)

PH[1] is a GPIO pin. eMIOS[30] is an input/output channel pin for the eMIOS200 module. FEC_RX_DV the Ethernet receive data valid input pin.

3.4.8.3 PH2 — GPIO (PH[2]) / eMIOS Channel (eMIOS[29]) / Ethernet Transmit Enable (FEC_TX_EN)

PH[2] is a GPIO pin. eMIOS[29] is an input/output channel pin for the eMIOS200 module. FEC_TX_EN is the Ethernet transmit enable output pin.

3.4.8.4 PH3 — GPIO (PH[3]) / eMIOS Channel (eMIOS[28]) / Ethernet Receive Error (FEC_RX_ER)

PH[3] is a GPIO pin. eMIOS[28] is an input/output channel pin for the eMIOS200 module. FEC_RX_ER the Ethernet receive error input pin.

3.4.8.5 PH4 — GPIO (PH[4]) / eMIOS Channel (eMIOS[27]) / Ethernet Receive Data (FEC_RXD[0])

PH[4] is a GPIO pin. eMIOS[27] is an input/output channel pin for the eMIOS200 module. FEC_RXD[0] is an Ethernet receive data input pin.

3.4.8.6 PH5 — GPIO (PH[5]) / eMIOS Channel (eMIOS[26]) / Ethernet Receive Data (FEC_RXD[1])

PH[5] is a GPIO pin. eMIOS[26] is an input/output channel pin for the eMIOS200 module. FEC_RXD[1] is an Ethernet receive data input pin.

3.4.8.7 PH6 — GPIO (PH[6]) / eMIOS Channel (eMIOS[25]) / Ethernet Receive Data (FEC_RXD[2])

PH[6] is a GPIO pin. eMIOS[25] is an input/output channel pin for the eMIOS200 module. FEC_RXD[2] is an Ethernet receive data input pin.

3.4.8.8 PH7 — GPIO (PH[7]) / eMIOS Channel (eMIOS[24]) / Ethernet Receive Data (FEC_RXD[3])

PH[7] is a GPIO pin. eMIOS[24] is an input/output channel pin for the eMIOS200 module. FEC_RXD[3] is an Ethernet receive data input pin.

3.4.8.9 PH8 — GPIO (PH[8]) / eMIOS Channel (eMIOS[23])

PH[8] is a GPIO pin. eMIOS[23] is an input/output channel pin for the eMIOS200 module.

3.4.8.10 PH9 — GPIO (PH[9]) / eMIOS Channel (eMIOS[22])

PH[9] is a GPIO pin. eMIOS[22] is an input/output channel pin for the eMIOS200 module.

3.4.8.11 PH10 — GPIO (PH[10]) / eMIOS Channel (eMIOS[21])

PH[10] is a GPIO pin. eMIOS[21] is an input/output channel pin for the eMIOS200 module.

3.4.8.12 PH11 — GPIO (PH[11]) / eMIOS Channel (eMIOS[20])

PH[11] is a GPIO pin. eMIOS[20] is an input/output channel pin for the eMIOS200 module.

3.4.8.13 PH12 — GPIO (PH[12]) / eMIOS Channel (eMIOS[19])

PH[12] is a GPIO pin. eMIOS[19] is an input/output channel pin for the eMIOS200 module.

3.4.8.14 PH13 — GPIO (PH[13]) / eMIOS Channel (eMIOS[18])

PH[13] is a GPIO pin. eMIOS[18] is an input/output channel pin for the eMIOS200 module.

3.4.8.15 PH14 — GPIO (PH[14]) / eMIOS Channel (eMIOS[17])

PH[14] is a GPIO pin. eMIOS[17] is an input/output channel pin for the eMIOS200 module.

3.4.8.16 PH15 — GPIO (PH[15]) / eMIOS Channel (eMIOS[16])

PH[15] is a GPIO pin. eMIOS[16] is an input/output channel pin for the eMIOS200 module.

3.4.9 Port J Pins

3.4.9.1 PJ0 — GPIO (PJ[0]) / eMIOS Channel (eMIOS[15]) / DSPI_A Peripheral Chip Select (PCS_A[4])

PJ[0] is a GPIO pin. eMIOS[15] is an input/output channel pin for the eMIOS200 module. PCS_A[4] is a peripheral chip select output pin for the DSPI A module.

3.4.9.2 PJ1 — GPIO (PJ[1]) / eMIOS Channel (eMIOS[14]) / DSPI_A Peripheral Chip Select (PCS_A[5])

PJ[1] is a GPIO pin. eMIOS[14] is an input/output channel pin for the eMIOS200 module. PCS_A[5] is a peripheral chip select output pin for the DSPI A module.

3.4.9.3 PJ2 — GPIO (PJ[2]) / eMIOS Channel (eMIOS[13]) / DSPI_B Peripheral Chip Select (PCS_B[1])

PJ[2] is a GPIO pin. eMIOS[13] is an input/output channel pin for the eMIOS200 module. PCS_B[1] is a peripheral chip select output pin for the DSPI B module.

3.4.9.4 PJ3 — GPIO (PJ[3]) / eMIOS Channel (eMIOS[12]) / DSPI_B Peripheral Chip Select (PCS_B[2])

PJ[3] is a GPIO pin. eMIOS[12] is an input/output channel pin for the eMIOS200 module. PCS_B[2] is a peripheral chip select output pin for the DSPI B module.

3.4.9.5 PJ4 — GPIO (PJ[4]) / eMIOS Channel (eMIOS[11]) / DSPI_C Peripheral Chip Select (PCS_C[3])

PJ[4] is a GPIO pin. eMIOS[11] is an input/output channel pin for the eMIOS200 module. PCS_C[3] is a peripheral chip select output pin for the DSPI C module.

3.4.9.6 PJ5 — GPIO (PJ[5]) / eMIOS Channel (eMIOS[10]) / DSPI_C Peripheral Chip Select (PCS_C[4])

PJ[5] is a GPIO pin. eMIOS[10] is an input/output channel pin for the eMIOS200 module. PCS_C[4] is a peripheral chip select output pin for the DSPI C module.

3.4.9.7 PJ6 — GPIO (PJ[6]) / eMIOS Channel (eMIOS[9]) / DSPI_D Peripheral Chip Select (PCS_D[5])

PJ[6] is a GPIO pin. eMIOS[9] is an input/output channel pin for the eMIOS200 module. PCS_D[5] is a peripheral chip select output pin for the DSPI D module.

3.4.9.8 PJ7 — GPIO (PJ[7]) / eMIOS Channel (eMIOS[8]) / DSPI_D Peripheral Chip Select (PCS_D[1])

PJ[7] is a GPIO pin. eMIOS[8] is an input/output channel pin for the eMIOS200 module. PCS_D[1] is a peripheral chip select output pin for the DSPI D module.

3.4.9.9 PJ8 — GPIO (PJ[8]) / eMIOS Channel (eMIOS[7])

PJ[8] is a GPIO pin. eMIOS[7] is an input/output channel pin for the eMIOS200 module. PJ[8] can be configured as a wakeup pin in the CRP_PWKENH register.

3.4.9.10 PJ9 — GPIO (PJ[9]) / eMIOS Channel (eMIOS[6])

PJ[9] is a GPIO pin. eMIOS[6] is an input/output channel pin for the eMIOS200 module. PJ[9] can be configured as a wakeup pin in the CRP_PWKENH register.

3.4.9.11 PJ10 — GPIO (PJ[10]) / eMIOS Channel (eMIOS[5])

PJ[10] is a GPIO pin. eMIOS[5] is an input/output channel pin for the eMIOS200 module. PJ[10] can be configured as a wakeup pin in the CRP_PWKENH register.

3.4.9.12 PJ11 — GPIO (PJ[11]) / eMIOS Channel (eMIOS[4])

PJ[11] is a GPIO pin. eMIOS[4] is an input/output channel pin for the eMIOS200 module. PJ[11] can be configured as a wakeup pin in the CRP_PWKENH register.

3.4.9.13 PJ12 — GPIO (PJ[12]) / eMIOS Channel (eMIOS[3])

PJ[12] is a GPIO pin. eMIOS[3] is an input/output channel pin for the eMIOS200 module. PJ[12] can be configured as a wakeup pin in the CRP_PWKENH register.

3.4.9.14 PJ13 — GPIO (PJ[13]) / eMIOS Channel (eMIOS[2])

PJ[13] is a GPIO pin. eMIOS[2] is an input/output channel pin for the eMIOS200 module. PJ[13] can be configured as a wakeup pin in the CRP_PWKENH register.

3.4.9.15 PJ14 — GPIO (PJ[14]) / eMIOS Channel (eMIOS[1])

PJ[14] is a GPIO pin. eMIOS[1] is an input/output channel pin for the eMIOS200 module. PJ[14] can be configured as a wakeup pin in the CRP_PWKENH register.

3.4.9.16 PJ15 — GPIO (PJ[15]) / eMIOS Channel (eMIOS[0])

PJ[15] is a GPIO pin. eMIOS[0] is an input/output channel pin for the eMIOS200 module.

3.4.10 Port K Pins

3.4.10.1 PK0 — GPIO (PK[0]) / Media Local Bus Clock (MLBCLK) / DSPI_B Clock (SCK_B) / Clock Output (CLKOUT)

PK[0] is a GPIO pin. MLBCLK is the Media Local Bus (MLB) clock input pin. SCK_B is the SPI clock pin for the DSPI B module. CLKOUT is the external bus interface clock output (test mode only).

3.4.10.2 PK1 — GPIO (PK[1]) / Media Local Bus Signal (MLBSIG) / DSPI_B Data Output (SOUT_B) / DSPI_D Peripheral Chip Select (PCS_D[4])

PK[1] is a GPIO pin. MLBSIG is the bidirectional signal line that transfers bus management data to/from the MOST network controller. SOUT_B is the data output pin for the DSPI B module. PCS_D[4] is a peripheral chip select output pin for the DSPI D module.

3.4.10.3 PK2 — GPIO (PK[2]) / Media Local Bus Data (MLBDAT) / DSPI_B Data Input (SIN_B) / DSPI_D Peripheral Chip Select (PCS_D[5])

PK[2] is a GPIO pin. MLBDAT is the bidirectional data line that transfers user data to/from the MOST network controller. SIN_B is the data input pin for the DSPI B module. PCS_D[5] is a peripheral chip select output pin for the DSPI D module.

3.4.10.4 PK3 — GPIO (PK[3]) / FlexRay Channel A Receive (FR_A_RX) / External Analog Mux Address Output (MA[0]) / DSPI_C Peripheral Chip Select (PCS_C[1])

PK[3] is a GPIO pin. FR_A_RX in the FlexRay Channel A receive pin. MA[0] is an address output for an external analog multiplexer used to select the multiplexer input channel to connect to the ADC. PCS_C[1] is a peripheral chip select output pin for the DSPI C module. PK[3] can be configured as a wakeup pin in the CRP_PWKENH register.

3.4.10.5 PK4 — GPIO (PK[4]) / FlexRay Channel A Transmit (FR_A_TX) / External Analog Mux Address Output (MA[1]) / DSPI_C Peripheral Chip Select (PCS_C[2])

PK[4] is a GPIO pin. FR_A_TX is the FlexRay Channel A transmit pin. MA[1] is an address output for an external analog multiplexer used to select the multiplexer input channel to connect to the ADC. PCS_C[2] is a peripheral chip select output pin for the DSPI C module.

3.4.10.6 PK5 — GPIO (PK[5]) / FlexRay Channel A Transmit Enable (FR_A_TX_EN) / External Analog Mux Address Output (MA[2]) / DSPI_C Peripheral Chip Select (PCS_C[3])

PK[5] is a GPIO pin. FR_A_TX_EN in the FlexRay Channel A transmit enable pin. MA[2] is an address output for an external analog multiplexer used to select the multiplexer input channel to connect to the ADC. PCS_C[3] is a peripheral chip select output pin for the DSPI C module.

3.4.10.7 PK6 — GPIO (PK[6]) / FlexRay Channel B Receive (FR_B_RX) / DSPI_B Peripheral Chip Select (PCS_B[1]) / DSPI_C Peripheral Chip Select (PCS_C[4])

PK[6] is a GPIO pin. FR_B_RX in the FlexRay Channel B receive pin. PCS_B[1] is a peripheral chip select output pin for the DSPI B module. PCS_C[4] is a peripheral chip select output pin for the DSPI C module. PK[6] can be configured as a wakeup pin in the CRP_PWKENH register.

3.4.10.8 PK7 — GPIO (PK[7]) / FlexRay Channel B Transmit (FR_B_TX) / DSPI_B Peripheral Chip Select (PCS_B[2]) / DSPI_C Peripheral Chip Select (PCS_C[5])

PK[7] is a GPIO pin. FR_B_TX is the FlexRay Channel B transmit pin. PCS_B[2] is a peripheral chip select output pin for the DSPI B module. PCS_C[5] is a peripheral chip select output pin for the DSPI C module.

3.4.10.9 PK8 — GPIO (PK[8]) / FlexRay Channel B Transmit Enable (FR_B_TX_EN) / DSPI_B Peripheral Chip Select (PCS_B[3]) / DSPI_A Peripheral Chip Select (PCS_A[1])

PK[8] is a GPIO pin. FR_B_TX_EN in the FlexRay Channel B transmit enable pin. PCS_B[3] is a peripheral chip select output pin for the DSPI B module. PCS_A[1] is a peripheral chip select output pin for the DSPI A module.

3.4.10.10 PK9 — GPIO (PK[9]) / Clock Output (CLKOUT) / DSPI_D Peripheral Chip Select (PCS_D[1]) / DSPI_A Peripheral Chip Select (PCS_A[2]) / Boot Configuration (BOOTCFG)

PK[9] is a GPIO pin. CLKOUT is the external bus interface clock output (user mode only). PCS_D[1] is a peripheral chip select output pin for the DSPI D module. PCS_A[2] is a peripheral chip select output pin for the DSPI A module. During reset, BOOTCFG is an input to control certain boot options.

3.4.10.11 PK10 — GPIO (PK[10]) / DSPI_B Peripheral Chip Select (PCS_B[5]) / DSPI_D Peripheral Chip Select (PCS_D[2]) / DSPI_A Peripheral Chip Select (PCS_A[3])

PK[10] is a GPIO pin. PCS_B[5] is a peripheral chip select output pin for the DSPI A module. PCS_D[2] is a peripheral chip select output pin for the DSPI A module. PCS_A[3] is a peripheral chip select output pin for the DSPI A module.

3.4.11 Nexus Signals

Except as noted, the Nexus signals are available only on the 256-pin package. For more information, see Chapter 36, Nexus Development Interface (NDI).

3.4.11.1 Nexus Event In EVTI

$\overline{\text{EVTI}}$ is an input that is read during a debug port reset to enable or disable the Nexus Auxiliary port for data trace. After reset, the $\overline{\text{EVTI}}$ pin is used to initiate program and data trace synchronization messages or generate a breakpoint. On the 208-pin BGA package, this pin is multiplexed with PC2.

3.4.11.2 Nexus Event Out EVTO

$\overline{\text{EVTO}}$ is an output that provides timing to a development tool for a single watchpoint or breakpoint occurrence. On the 208-pin BGA package, this pin is multiplexed with PC3.

3.4.11.3 Nexus Message Clock Out MCKO

MCKO is a free running clock output to the development tools which is used for timing of the MDO and $\overline{\text{MSEO}}$ signals.

3.4.11.4 Nexus Message Data Out MDO[0]

MDO[0] indicates power-on reset (POR) status. In addition, MDO[0] indicates the lock status of the system clock following a POR. MDO[0] is driven high following a POR until the system clock achieves lock, at which time it is then negated. There is an internal pullup on MDO[0].

3.4.11.5 Nexus Message Data Out MDO[11:1]

MDO[11:1] are the trace message outputs to the development tools for full port mode.

3.4.11.6 Nexus Message Start/End Out MSEO[1:0]

MSEO[1:0] are outputs that indicate when messages start and end on the MDO pins.

3.4.12 Reset and Configuration Signals

3.4.12.1 External Reset Input RESET

The RESET pin is a bidirectional I/O pin. It is asserted by an external device to reset the all modules of the device MCU. It is also an open drain output signal that is asserted during an internal reset. For more information, see Chapter 4, Resets.

3.4.13 JTAG Signals

For more information, see Chapter 35, IEEE 1149.1 Test Access Port Controller (JTAGC).

3.4.13.1 JTAG Test Clock Input TCK

TCK provides the clock input for the on-chip test logic.

3.4.13.2 JTAG Test Data Input TDI

TDI provides the serial test instruction and data input for the on-chip test logic.

3.4.13.3 JTAG Test Data Output TDO

TDO provides the serial test data output for the on-chip test logic.

3.4.13.4 JTAG Test Mode Select Input TMS

TMS controls test mode operations for the on-chip test logic.

3.4.13.5 JTAG Compliance Input JCOMP

The JCOMP pin is used to enable the JTAG TAP controller.

3.4.13.6 Test Mode Enable Input TEST

The TEST pin is used to place the chip in test mode. It must be tied to V_{SS} for normal operation.

3.4.14 Clock Synthesizer Signals

3.4.14.1 Crystal Oscillator Input / External Clock Input EXTAL

EXTAL is the input pin for an external crystal oscillator or an external clock source.

3.4.14.2 Crystal Oscillator Output XTAL

XTAL is the output pin for an external crystal oscillator.

3.4.14.3 System Clock Output CLKOUT

CLKOUT is the device system clock output. This signal is multiplexed with PK9 (user mode) and PK0 (test mode).

3.4.15 Power / Ground Signals

3.4.15.1 Internal Logic Supply Input V_{DD}

V_{DD} is the 1.2 V (nominal) internal logic supply input.

3.4.15.2 Fixed 3.3V Internal Supply Input V_{DD33}

V_{DD33} is the 3.3V (nominal) internal supply input.

3.4.15.3 Analog Supply V_{DDA}

V_{DDA} is the 3.3 or 5.0 V (nominal) analog supply input pin for the ADC.

3.4.15.4 External I/O Supply Input $V_{DDE}n$

$V_{DDE}n$ is the 3.3 or 5.0 V (nominal) external I/O supply input. Four separate inputs are provided.

3.4.15.5 Media Local Bus Supply Input V_{DDEMLB}

V_{DDEMLB} is the 2.5 – 3.3 V supply input for the Media Local Bus interface.

3.4.15.6 Nexus Interface Supply Input V_{DDENEX}

V_{DDENEX} is the 3.3 V (nominal) supply input for the Nexus Debug Interface. This supply is used only on the 256-pin package.

3.4.15.7 Clock Synthesizer Power Input V_{DDSYN}

V_{DDSYN} is the is the power supply input for the FMPLL.

3.4.15.8 Voltage Regulator Control Voltage V_{RC}

V_{RC} is the 3.3 V or 5 V (nominal) input power supply.

3.4.15.9 Voltage Regulator Control Output V_{RCCTL}

V_{RCCTL} is the current control for external NPN transistor.

3.4.15.10 Supply V_{RCSEL}

V_{RCSEL} is the input power supply range select.

- V_{RCSEL} = 0: 3.3 V supplied external (3.3 V mode)
- V_{RCSEL} = 1: 3.3 V internal voltage regulator (5 V mode)

3.4.15.11 Analog High Voltage Reference V_{RH}

V_{RH} is the analog high voltage reference.

3.4.15.12 Analog Low Voltage Reference V_{RL}

V_{RL} is the analog low voltage reference.

3.4.15.13 Ground V_{SS}

V_{SS} is the ground reference input.

3.4.15.14 Analog Ground V_{SSA}

V_{SSA} is the analog ground reference input.

3.4.15.15 Clock Synthesizer Ground Input V_{SSSYN}

V_{SSSYN} is the ground reference input for the FMPLL clock synthesizer.

NOTE

If VDDEMLB is greater than VDD33, then the PK0-2 pins will have extra leakage current if the pin is low (either output drive low, external drive low, or internal pull-down). Typical leakage currents at room temperature is 0.1µA per pin for a differential voltage of 0.18V, 1µA per per for a differential voltage of 0.27V, 5µA per pin for a differential voltage of 0.33V up to 300µA per pin for a differential voltage of 0.5V. If the internal pull devices are enabled, then the extra current will add to the make the internal pullup stronger and will weaken the internal pulldown. Note that the Nexus pins in the 256MAPBGA development package also have this issue if VDDENEX is greater than VDD33, but there is no issue in the 208MAPBGA package since VDDENEX is grounded. In order to avoid this, design a circuitry which will allow for extra leakage if there is the possibility that the VDDEMLB or VDDENEX supply can be greater than VDD33.

Chapter 4
Resets

4.1 Introduction

This chapter describes the resets and reset sources for the PXN20.

The reset sources supported in the PXN20 are:

- Power-on reset (POR)
- Low-voltage inhibit (LVI) reset
- External reset
- Loss-of-lock reset
- Loss-of-clock reset
- Watchdog timer reset
- JTAG reset
- Checkstop reset (both Z6 and Z0 cores)
- Software-system reset

All reset sources are processed by the reset controller, which is located in the SIU module (Chapter 8, System Integration Unit (SIU)). The reset controller monitors the reset input sources. Upon detection of a reset event, the reset controller resets internal logic and controls the assertion of the $\overline{\text{RESET}}$ pin.

The MCU is clocked by the 16 MHz IRC clock after any reset.

The reset status register (SIU_RSR) gives the source, or sources, of the last reset and is updated for all reset sources except JTAG reset

The BOOTCFG pin controls the MCU boot sequence after any reset. If the pin is driven low during the MCU reset, the MCU boots from internal flash and the Reset Configuration Halfword (RCHW) controls the boot sequence. The RCHW needs to be programmed by user in internal flash in one of predefined locations together with the user application start address.

If the pin is driven high, the BAM executes the serial boot sequence.

See Chapter 9, Boot Assist Module (BAM), for more details about the boot procedures.

4.2 External Signal Description

Refer to Table 3-1 and Section 3.4, Detailed Signal Description, for signal properties.

4.2.1 Reset ($\overline{\text{RESET}}$)

This pin provides the system reset. It is an open-drain, active-low bidirectional pin. It acts as an input to initialize the MCU to a known start-up state, and an output when an internal MCU function causes a reset. Externally asserting the $\overline{\text{RESET}}$ pin resets the chip asynchronously. The chip remains in reset as long as the external $\overline{\text{RESET}}$ pin is asserted. Any internal reset event asserts the $\overline{\text{RESET}}$ pin for as long as the reset event is active. When the internal reset sources are negated, the $\overline{\text{RESET}}$ pin is asserted by the reset controller for 1000 clocks (16 clocks if CRP_RECPRTR[FASTREC] = 1). Then the reset controller stops asserting the $\overline{\text{RESET}}$ pin. After another predefined time, the $\overline{\text{RESET}}$ pin is sampled, and if still asserted then an external reset request is assumed. When the $\overline{\text{RESET}}$ pin is sampled high (the pin is no longer being driven low by the PXN20 reset logic or by external logic that might be requesting reset), the BOOTCFG reset configuration pin (pin PK9) is sampled and the internal reset to the chip negates.

On assertion, the SIU_RSR[ERS] flag is set.

4.2.2 Boot Configuration (BOOTCFG)

The BOOTCFG pin (pin name PK9 in package diagrams and signal lists) is used to determine the boot mode initiated by the BAM program. The pin state during reset is latched in the SIU_RSR[BOOTCFG] field. The BAM program uses the BOOTCFG field to determine whether initiate internal flash boot mode or a CAN or SCI “serial” boot.

Refer to 8.3.2.2, Reset Status Register (SIU_RSR), for more information.

NOTE

The reset controller latches the state of the BOOTCFG pin into the SIU_RSR register 4 clock cycles prior to the negation of $\overline{\text{RESET}}$.

4.3 Functional Description

4.3.1 Z6, Z0 Cores Reset Vectors

The reset vectors for the Z6 and Z0 cores in the PXN20 MCU are controlled via the Z6VEC and Z0VEC registers in the Clock, Reset, and Power control (CRP) module. The power-on reset values for the Z6VEC and Z0VEC registers point to the first instruction the BAM program.

The Z0 core is disabled after the POR and Z6 is active. Thus, following the POR, the Z6 core starts to execute the BAM code. See Chapter 9, Boot Assist Module (BAM), for more details about the boot process.

4.3.2 Reset Sources

4.3.2.1 Power-on Reset (POR)

The internal Power On Reset (POR) monitors the main supply input voltage (V_{DDA}) and shall not release the internal reset line until V_{DDA} is above the de-assertion threshold. On assertion, the SIU_RSR[PORS] flag is set.

4.3.2.2 Low-Voltage Inhibit (LVI) Resets

The internal LVI reset signals are asserted when the voltage on the corresponding supply is below defined values. The following are the LVI resets:

- LVI12—LVI on internal 1.2 V supply (V_{DD})
- LVI33—LVI on internal 3.3 V supply to I/O pads and flash (V_{DD33})
- LVISYN—LVI on 3.3 V V_{DDSYN} supply
- LVIL_VDDA—LVI on V_{DDA} input supply and generates reset
- LVI_VDDA—LVI on V_{DDA} input supply and generates either a reset or an interrupt
  - Reset: configured in the CRP_SOCSC register, reported in SIU_RSR register (default)
  - Interrupt: configured and reported in CRP_SOCSC register

On assertion, the SIU_RSR[PORS] flag is set.

4.3.2.3 External Reset

When the reset controller detects assertion of the $\overline{\text{RESET}}$ pin, the internal reset signal is asserted. The SIU_RSR[ERS] bit is set, and all other reset status bits in the SIU_RSR are cleared.

4.3.2.4 Loss-of-Lock Reset

A loss-of-lock reset occurs when the PLL loses lock and the loss-of-lock reset enable (LOLRE) bit in the PLL enhanced synthesizer control register 2 (ESYNCR2) is set. The internal reset signal and $\overline{\text{RESET}}$ pin are asserted. The SIU_RSR[LLRS] bit is set, and all other reset status bits in the SIU_RSR are cleared.

4.3.2.5 Loss-of-Clock Reset

A loss-of-clock reset occurs when a failure is detected in either the reference clock signal or PLL output when the PLL is enabled. The internal reset signal and $\overline{\text{RESET}}$ pin are asserted. The SIU_RSR[LCRS] bit is set, and all other reset status bits in the SIU_RSR are cleared.

4.3.2.6 Watchdog Timer Reset

A watchdog timer reset occurs when the SWT watchdog timer is enabled and a timeout occurs. The internal reset signal and $\overline{\text{RESET}}$ pin are asserted. The SIU_RSR[WTRS] bit is set, and all other reset status bits in the SIU_RSR are cleared.

4.3.2.7 Z6 Core Checkstop Reset

When the Z6 core enters a checkstop state, and the checkstop reset is enabled (SIU_SRCR[CRE0] bit), a checkstop reset occurs. The internal reset signal and $\overline{\text{RESET}}$ pin are asserted. The SIU_RSR[CRS] bit is set and all other reset status bits in the SIU_RSR are cleared.

4.3.2.8 Z0 Core Checkstop Reset

When the Z0 core enters a checkstop state, and the checkstop reset is enabled (SIU_SRCR[CRE1] bit), a checkstop reset occurs. The internal reset signal and RESET pin are asserted. The SIU_RSR[CRS] bit is set and all other reset status bits in the SIU_RSR are cleared.

4.3.2.9 JTAG Reset

A system reset occurs when JTAG is enabled and the EXTEST, CLAMP, or HIGHZ instruction is executed by the JTAG controller. The internal reset signal is asserted. The state of the RESET pin is determined by the JTAG instruction. The reset status bits in the SIU_RSR are unaffected by JTAG reset.

4.3.2.10 Software System Reset

A software system reset is caused by writing to the SIU_RCR[SSR] bit. Setting the SSR bit causes an internal reset of the MCU. The internal reset signal and RESET pin are asserted. The SIU_RSR[SSRS] bit is set, and all other reset status bits in the SIU_RSR are cleared.

4.4 Reset Configuration

The reset state of the system is:

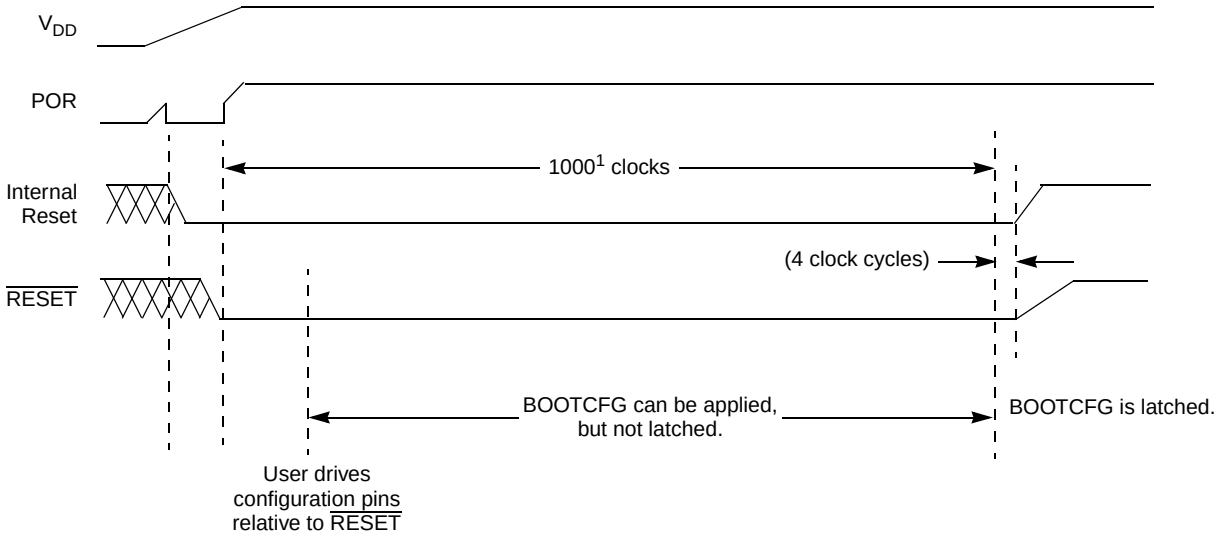
- All pads on ports A–K are placed in a disabled mode with output enables, input enables, and pull devices all disabled. PK9 is configured for BOOTCFG function.
- TDI pad is an input with pullup enabled.
- TDO pad is an output with fastest slew rate selected.
- TCK pad is an input with pulldown enabled.
- TMS pad is an input with pullup enabled.
- JCOMP pad is an input with pulldown enabled.
- RESET pin is configured as open drain output with pullup disabled and initially driven low, but switched to an input with pullup enabled after the reset sequence.
- BOOTCFG pin is an input, the pin data is latched 4 clock cycles before the RESET signal is negated (high).
- Nexus pads
  - The following configuration is valid as long as the NPC is out of reset and enabled via JTAG:
    - EVTO output with fastest slew rate enabled, high
    - EVTI output buffer disabled, input buffer enabled, pullup enabled
    - MCKO output with fastest slew rate enabled. Low until MCKO_EN = 1
    - MDO[11:0] output with fastest slew rate enabled, low
    - MSEO[1:0] output with fastest slew rate enabled, high

NOTE

Nexus is only available on the 256MAPBGA emulation package.

4.4.1 Reset Configuration Timing

The timing diagram in Figure 4-1 shows the sampling of the BOOTCFG (PK9) pin for a power-on reset. The timing diagram is also valid for internal/external resets assuming V_{DD} and V_{DD33} are within valid operating ranges. The value of the BOOTCFG pin is latched 4 clock cycles before the negation of the RESET pin and stored in the reset status register.



1 If the CRP_RECPTR[FASTREC] is set, then the clock count is 16 for Sleep mode recovery.

Figure 4-1. Reset Configuration Timing

Chapter 5
System Clock Description

5.1 Introduction

This chapter describes the clock architecture and sources of the PXN20 system clocks.

The PXN20 has a number of different clock sources, serving various application requirements and allowing maximum flexibility for the user application.

These are:

- High-frequency crystal oscillator described in Section 5.1.3, External High-Frequency Crystal (4 – 40 MHz XTAL), supporting external crystals in the range of 4 – 40MHz. This is mainly used as a precise clock source and PLL input clock source.
- Fast on-chip RC oscillator, described in Section 5.1.4, Internal High-Frequency RC Oscillator (16 MHz_IRC). This is mainly used as the default clock source with fast startup, a fast clock source in low-power modes, and for the watchdog timer.
- Slow on-chip RC oscillator, described in Section 5.1.5, Internal Low-Frequency RC Oscillator (128 kHz_IRC). This is mainly used as an independent clock source for the ultra low power modes.
- 32 kHz crystal oscillator described in Section 5.1.6, External Low-Frequency Crystal (32 kHz_XTAL), supporting an external crystal of 32 kHz. This is used for the Real Time Clock applications and alternate clock source to the slow on-chip RC oscillator.
- Phase-locked loop, described in Section 5.1.7, FMPLL, supporting spread spectrum modulation to reduce EMI and providing system frequencies above 40 MHz. This is mainly used in full run mode and uses the high frequency crystal as its clock source. See Chapter 7, Frequency Modulated Phase-Locked Loop (FMPLL), for more information.

5.1.1 Features

The following list summarizes the system clock and clock generation on the PXN20:

- System clock can be derived from the following sources
  - 4 – 40 MHz XTAL
  - FMPLL
  - 16 MHz IRC oscillator
- Programmable output clock divider of system clock (÷1, ÷2, ÷4, ÷8, ÷16)
- Separate programmable peripheral bus clock divider ratio (÷1, ÷2, ÷4, ÷8) applied to system clock
- Frequency Modulated Phase-locked loop (FMPLL)
  - Input clock frequency from 4 MHz to 40 MHz

  - Clock source from external oscillator
  - Lock detect circuitry continuously monitors lock status
  - Loss of clock (LOC) detection for reference and feedback clocks
  - On-chip loop filter (for improved electromagnetic interference performance and reduces number of external components required)
  - Programmable frequency modulation
- On-chip crystal oscillator (4 – 40 MHz XTAL) supporting 4 MHz to 40 MHz crystals
- Dedicated 16 MHz internal RC oscillator
  - 16 MHz internal RC oscillator supports low speed code execution and clocking of peripherals through selection as the system clock
  - Used as default clock source out of reset
  - Provides a clock for rapid start-up from low power modes
  - Provides a back-up clock in the event of PLL or external oscillator clock failure
  - Provides watchdog timer
  - 5% accuracy over the operating temperature range (after factory trim)
  - Trimming registers to support frequency adjustment with in-application calibration
- Dedicated internal 128 kHz_IRC oscillator for low power mode operation and self wake-up
  - 5% accuracy (after factory trim)
  - Trimming registers to support improve accuracy with in-application calibration
- Dedicated 32 kHz_XTAL external oscillator for accurate timed wake-up

5.1.2 Clock Sources

The PXN20 clock sources are:

- High Frequency Crystal oscillator (4 – 40 MHz XTAL)
- Fast on-chip RC oscillator (16 MHz_IRC)
- Slow on-chip RC oscillator (128 kHz_IRC)
- 32 kHz crystal oscillator (32 kHz_XTAL)
- Phase-Locked Loop (FMPLL)

The clock sources available on the PXN20 are shown in Figure 5-1 and discussed in more detail in subsequent sections.

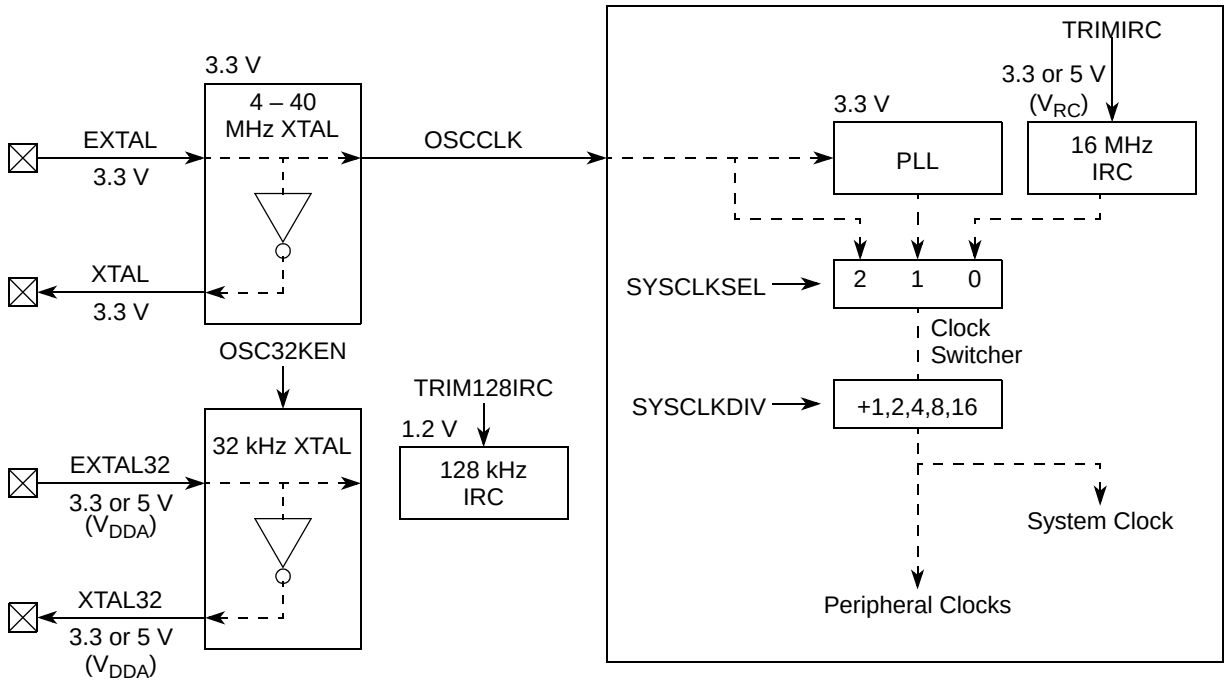


Figure 5-1. PXN20 Available Clock Sources

5.1.3 External High-Frequency Crystal (4 – 40 MHz XTAL)

The 4 – 40 MHz XTAL clock uses an external high frequency crystal to provide the main clock source to the circuit. The on-chip crystal oscillator has automatic level control and supports 4 MHz to 40 MHz crystals. It can be used as external square-wave input.

A low power output buffer is selected automatically in Sleep mode to allow crystals of 8 MHz and below to continue to run and clock the Real Time Clock (RTC) and Autonomous Periodic Interrupts (API).

A crystal in the full 4 – 40 MHz range can be kept enabled in Sleep mode to facilitate fast start up. In this mode, the crystal oscillator cannot drive any internal modules (e.g.,RTC/ API), as the output driver is disabled.

5.1.3.1 4 – 40 MHz XTAL Features

- Input frequency range: 4 MHz – 40 MHz
- Automatic level control
- Oscillator input mode 3.3V (V_{DDSYN})
- External crystal mode enabled by default after reset. External square wave input supported, but crystal bypass mode should be selected in startup code if an external crystal is not used.
- FMPLL reference clock
- Jitter < 0.5%, over one CAN frame
- Duty cycle: 40 – 60%
- Clock source capable of supporting FlexRay communications

  - jitter < 0.5%
  - duty cycle: 50% ±10%
- Clock source can be kept alive in Sleep mode to facilitate fast start up. However, if > 8 MHz is required, it cannot drive the RTC/API as the output driver needs to be disabled.
- Can be stopped in low power mode to reduce current consumption
- PLL Off mode to reduce consumption with external clock
- External square-wave input with PLL Off mode XTAL enabled by default out of Reset, turned off for Square wave i/p

5.1.4 Internal High-Frequency RC Oscillator (16 MHz_IRC)

This internal RC oscillator provides a 16 MHz clock source to the device. The initial trimming value is copied into the trimming register during power on reset (POR). The device's individual value is determined during one of the silicon test steps and stored in the device test Flash block. Until the Flash is up and stable, the IRC runs at its untrimmed frequency.

5.1.4.1 16 MHz_IRC Features

- 16 MHz_IRC is the default system clock out of reset. This clock features the following:
  - Clock for rapid startup from low power modes
  - Can be configured to run in Sleep mode
  - Backup clock in case of external oscillator or PLL failure
- Fast stabilization, enabling fast recovery
- Frequency trimmable for accuracy
- Powered from V_{RC} (3.3 V or 5 V nominal)
- Always enabled except in sleep modes when not being used (disabled in hardware only)
- The 16 MHz_IRC block contains no dividers of its own.

5.1.5 Internal Low-Frequency RC Oscillator (128 kHz_IRC)

The PXN20 includes a 128 kHz internal RC oscillator that is a highly reliable clock source during low-power modes. This internal RC oscillator provide a 128 kHz clock source to the device. The initial trimming value is copied into the trimming register during power on reset. This device's individual value is determined during one of the silicon test steps and stored in the device test Flash block. This trim value is retained through Sleep mode.

5.1.5.1 128 kHz_IRC Features

The 128 kHz_IRC clock features the following:

- Frequency trimmable for accuracy
- Option to clock the API to provide a wakeup
- Option to clock the RTC to provide timekeeping

- Powered from internal 1.2 V
- Optionally disabled in Sleep mode
- Current consumption < 2 µA
- 128 kHz ±35% across process, voltage and temperature (before trimming)
- 128 kHz ±10% across voltage and temperature (after factory trimming)
- 128 kHz ±2% across voltage and temperature for a minimum duration of 100 ms (after application trimming)

The internal 128 kHz RC oscillator is for low power mode operation and wake-up. It does not offer a system clock source.

The 128 kHz_IRC clock can be optionally stopped to reduce current consumption in Sleep mode. See Section 5.5.4, SWT Clock Domain.

5.1.6 External Low-Frequency Crystal (32 kHz_XTAL)

The PXN20 supports an external 32 kHz crystal oscillator to provide accurate wake-up and time-keeping functions. The 32 kHz_XTAL clock is an external low power, high accuracy clock source to provide the optional clock source for the RTC/API.

5.1.6.1 32 kHz_XTAL Features

- Two external pins required: EXTAL32 and XTAL32
- Supports external low frequency crystals in the range 32 kHz to 40 kHz
- Amplitude Level Control (ALC) to optimize power consumption
- Voltage and frequency filtering to guarantee clock frequency and stability
- Option to clock the API to provide a more accurate wakeup
- Option to clock the RTC to provide accurate time keeping
- Current consumption < 2 µA
- Powered from V_{DDA} (3.3 V or 5 V)
- Can be disabled via software

5.1.7 FMPLL

The FMPLL module is one of the clock sources for the system clock. It is used especially for higher speed run modes up to the maximum design speed of 128 MHz.[1] Its input clock is the undivided 4 – 40 MHz XTAL.

5.1.7.1 FMPLL Features

- Input clock frequency range: 4 MHz to 40 MHz (on EXTAL)

1.Maximum speed for the current revision of silicon may be lower. See the *PXN20 Microcontroller Data Sheet* for more information.

- Because the PXN20 uses a 16 MHz IRC as its default system clock, the FMPLL is put in PLL Off mode during reset, so that power dissipation is minimized by disabling the FMPLL until needed by the system.
- Programmable frequency multiplication factor settings generating VCO frequencies of 192 MHz – 600 MHz
- PLL Off mode (low-power mode)
- Register programmable output clock divider (ERFD)
- Programmable frequency modulation
  - Modulation applied as a triangle waveform
  - Peak-to-peak register programmable modulation depths of 0.5%, 1%, 1.5%, and 2% of the system frequency
  - Register programmable modulation rates of $F_{extal}/80$, $F_{extal}/40$, and $F_{extal}/20$
- Lock detect circuitry provides a signal indicating the FMPLL has acquired lock and continuously monitors the FMPLL output for any loss of lock
- Loss-of-clock circuitry monitors input reference and FMPLL output clocks with programmable ability to select a backup clock source as well as generate a reset or interrupt in the event of a failure
- PLL Analog can be turned off if not used.
- The FMPLL cannot run in Sleep mode

5.2 System Clock Architecture

The PXN20 clocking architecture is shown in Figure 5-2. The figure shows all clock sources that are available. It also shows clock selection and divider options that apply to each module. Peripheral sets are shown in Table 5-1.

To optimize system power consumption, the PXN20 supports both system- and peripheral-level clock dividers, and static clock gating using peripheral-level module disable (MDIS) bits and a system-level halt mechanism. Figure 5-2 shows the device-level clock gating mechanism for the PXN20. These features are detailed in Figure 5-3.

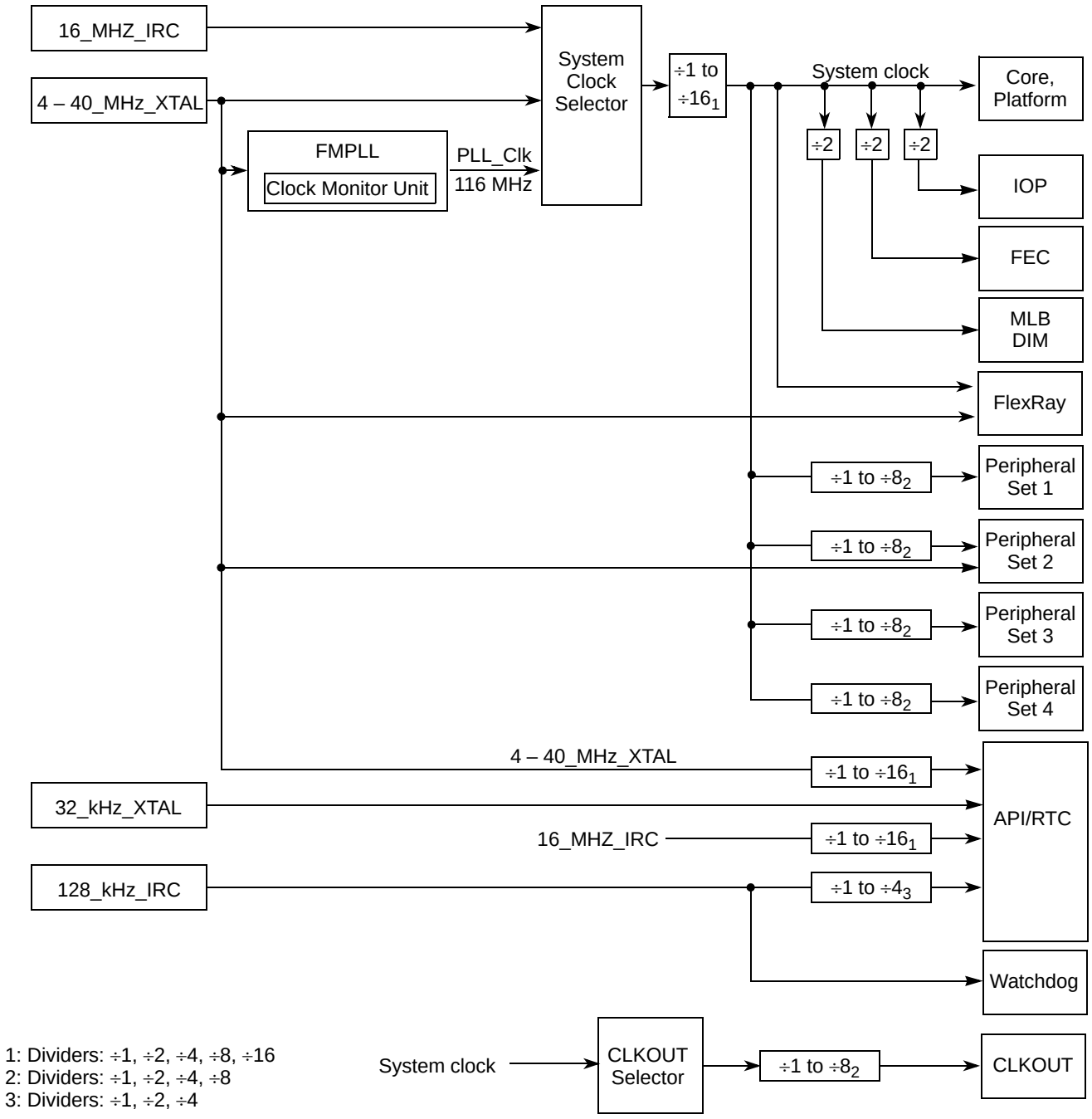


Figure 5-2. PXN20 System Clock Architecture

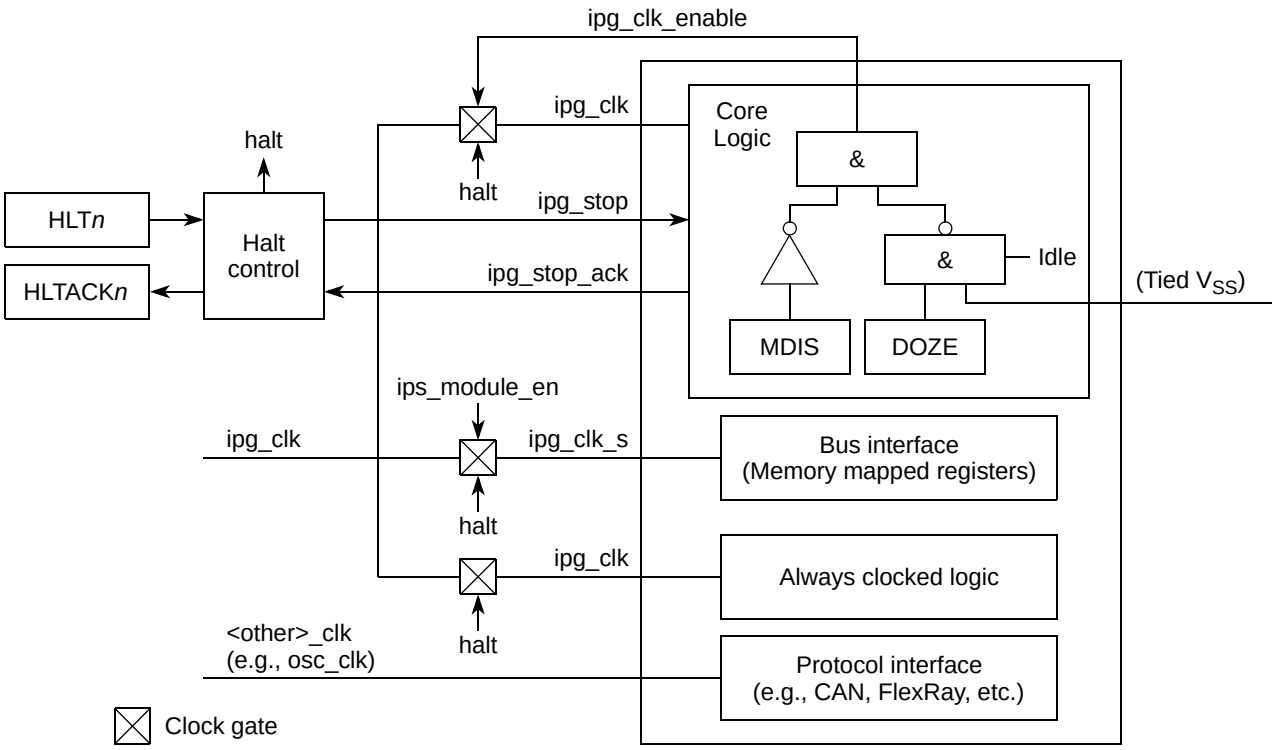


Figure 5-3. Detailed Clock Gating Scheme

5.3 Clock Dividers

5.3.1 System Clock Select

The source for the system clock can be selected by the SYSCLKSEL field of the SIU system clock register (SIU_SYSCLK) to be the 16 MHz IRC, the 4 – 40 MHz XTAL, or the FMPLL.

5.3.2 System Clock Dividers

The system clock dividers can be programmed to create a system clock, which is created from the selected clock source divided by 1, 2, 4, 8, or 16, based on the setting of the SYSCLKDIV field in the SIU system clock register (SIU_SYSCLK).

5.3.3 External Bus Clock (CLKOUT) Divider

The system clock divided by 1, 2, 4, or 8 based on the settings of the ECDF bit field in the SIU external clock control register (SIU_ECCR). The reset value of ECDF selects a CLKOUT frequency of one half of the system clock frequency.

NOTE

The CLKOUT provides a nominal 50% duty cycle clock with the exception of the case when the CLKOUT prescaler is equal to ÷ 1 and the system clock and has been divided by its prescaler. When running at this full speed (116 MHz system clock), the CLKOUT should be configured by the user with a divide ratio of at least ÷ 4.

Out of Reset the CLKOUT pin is disabled to minimize noise. This must be turned on by the user during initialization.

5.3.4 Nexus Message Clock (MCKO) Divider

The Nexus message clock (MCKO) divider can be programmed to divide the system clock by 1, 2, 4, or 8, based on the MCKO_DIV bit field in the port configuration register (PCR) in the Nexus port controller (NPC). The reset value of MCKO_DIV selects an MCKO clock frequency one half of the system clock frequency. The MCKO divider is configured by writing to the NPC through the JTAG port. The MCKO_EN bit may be used to disable the MCKO clock. The MCKO_GT bit may be used to disable the MCKO clock when Nexus is not actively transmitting messages on the Nexus port.

NOTE

The MCKO provides a nominal 50% duty cycle clock with the exception of the case that the MCKO prescaler is equal to ÷ 1 and the system clock has been divided by its prescaler. There is no guaranteed phase relationship between CLKOUT and MCKO.

NOTE

Z6 tracing is supported for MCKO divides of 1, 2, 4, or 8. Since the Z0 clocking is based on half the system bus clock frequency, the Z0 tracing is supported for MCKO divides of 2, 4, or 8 with a divide of 1 not supported. Also, concurrent tracing of both the Z6 and Z0 is supported for MCKO divides of 2, 4, or 8, with a divide of 1 not supported.

5.3.5 Peripheral Clock Dividers

The system and peripheral bus clocks can all be divided down to tune performance to meet the needs of the application, helping to save power and to meet the required peripheral speeds. The system clock speed can be divided down from ÷ 1 to ÷ 16 in discrete steps (÷ 1, ÷ 2, ÷ 4, ÷ 8, ÷ 16). This allows the CPU speed to be reduced to 1 MHz when operating from the 16 MHz IRC for slow dynamic run current. In addition the peripherals can divide this system bus clock speed for their normal operation from ÷ 1 to ÷ 8 in discrete steps (÷ 1, ÷ 2, ÷ 4, ÷ 8) enabling slow peripheral groups such as Peripheral Set 1 (Refer to Table 5-1 for more information) to be able to be optimized to run at more efficient speeds.

Table 5-1. Peripheral Sets

Peripheral Set 1	Peripheral Set 2	Peripheral Set 3	Peripheral Set 4
All eSCI modules	All FlexCAN modules	ADC	eMIOS
I^2C	All SPI modules	CTU	

NOTE

Unlisted peripherals such as the Flash, SIU, etc., are considered part of the Platform and hence are not listed here.

Each divider can be changed independently of the other dividers. However, the user must ensure that the system bus clock is always equal to or faster than the other clocks.

It should be noted that reducing clock speed for peripheral groups results in slower access times by the device masters and may impact performance of other operations as a result of restricting bandwidth on the respective IPS bus, or from the accessing master.

NOTE

- The ADC requires a 50% duty cycle clock. Thus, if the system clock has been divided by its prescaler, then the ÷ 2 clock divider internal to the ADC module must be selected (i.e., ADC MCR[ADCLKSEL] = 0).
- DMA operations are not supported for peripherals when the peripheral clock is divided.

NOTE

- If using the PIT to trigger the CTU, do not divide the CTU peripheral clock. SIU_SYSCLK[LPCLKDIV2] should be kept at the default setting SIU_SYSCLK[LPCLKDIV2]=0b00.

5.4 Software-Controlled Power Management

5.4.1 Module Disable (MDIS) Clock Gating

Static clock gating is enabled by software writes to configuration bits for the modules to disable the modules. Modules are re-enabled by software to ungate the module clocks.

The modules support software controlled clock gating where the application software can disable the non-memory-mapped portions of the blocks by writing to module disable (MDIS) bits in registers within the blocks. (The memory-mapped portions of the blocks are clocked by the system clock only when they are accessed.) The Nexus port controller (NPC) can be configured to disable the MCKO signal when there are no Nexus messages pending.

The flash array can be disabled by writing to the FDIS bit in the CRP module.

The modules that support software-controlled power management/clock gating are listed in Table 5-2 along with the registers and bits that disable each block. Default out of reset disables the software-controlled clocks.

Table 5-2. Software-Controlled Clock Gating Support

Block Name	Register Name	Bit Name
DSPI	DSPI_MCR Offset: Base + 0x0000	MDIS
ESCI	ESCI*x*_CR2 Offset: Base + 0x0004	MDIS
FlexCAN	CAN*x*_MCR Offset: Base + 0x0000	MDIS
EMIOS	EMIOS_MCR Offset: Base + 0x0000	MDIS
CTU	CTUPCR Offset: Base + 0x00CC	MDIS
PIT	PITCTRL Offset: Base + 0x0110	MDIS
I^2C	IBCR Offset: Base + 0x0002	MDIS
NPC	Nexus PCR[30]= MCKO_GT Nexus PCR[29]= MCKO_EN==MDIS Reg Index: 127	MCKO_EN, MCKO_GT

5.4.2 Halt Clock Gating

System clock gating is forced via the centralized halt mechanism. The SIU_HLT0 and SIU_HLT1 bits corresponding to individual modules are configured to determine which modules are clock gated.

The SIU_HLT0 and SIU_HLT1 bits are used to drive the stop inputs to the modules. After the module completes a clean shutdown, the module asserts the stop acknowledge handshake. The stop acknowledge is visible in the SIU_HLTACK0 and SIU_HLTACK1 read-only register bits. The modules are individually controlled and halted.

The halted module recovers when the HLT bit is cleared by software. After HLT is cleared, the device's logic re-enables the clocks to the modules and negates the stop signal after the required timing has been met.

There is no hardware disable for the eDMA and FlexRay modules. Thus before setting the HLT bits for these masters, software should take actions to prepare for the eDMA and FlexRay clocks to be stopped. Then software sets the HLT bits for the eDMA and FlexRay to indicate to the clock logic that the clocks to these modules can now be stopped.

When the Z0 and Z6 have executed WAIT instructions, then the clocks to the platform are also gated. The platform logic includes the MPU, AXBS, AIPS, and ECSM. The INTC and SIU are not clock gated to allow for an interrupt to be used to exit WAIT.

5.4.3 Core WAIT Clock Gating

Core clock gating is enabled via the CPU WAIT instruction.

The Z6 and Z0 cores may be idled by their WAIT instructions. The WAIT instructions are used as a power-saving feature to halt the core. Executing the WAIT instruction puts the corresponding core in an idle state at a clean transition point. When the core stops, clocks to the core are gated off, and the core asserts a signal indicating it is waiting for an interrupt. The state of this signal is software accessible via the appropriate SIU_HLTACK0 and SIU_HLTACK1 bits.

An interrupt to the corresponding core exits the WAIT instruction and the core continues to the appropriate interrupt service routine (ISR).

NOTE

If both the Z6 and Z0 cores are stopped (either in WAIT or disabled with ZxRST), then only an NMI interrupt will recover the core from WAIT. A core may be recovered from WAIT with either an NMI or external interrupt if the other core is not stopped.

5.5 Alternate Module Clock Domains

5.5.1 FlexCAN Clock Domains

The FlexCAN blocks have two distinct software-controlled clock domains. One of the clock domains is always derived from the system clock. This clock domain includes the message buffer logic. The source for the second clock domain can be the system clock or the 4 – 40 MHz XTAL output. The logic in the second clock domain controls the CAN interface pins. The CLK_SRC bit in the FlexCAN CTRL register selects between the system clock and the oscillator clock as the clock source for the second domain. Selecting the oscillator as the clock source ensures low jitter on the CAN bus. System software can gate both clocks by writing to the MDIS bit in the FlexCAN MCR register.

NOTE

To prevent improper FlexCAN behavior when switching of the system clock or the CAN protocol engine clock source, or before the desired clock source has stabilized, the FlexCAN module must first be disabled by setting the CAN*x*_MCR[MDIS] = 1.

If the oscillator clock source is selected, the frequency of the peripheral clock needs to be the same or greater than the oscillator clock frequency.

If the 4 – 40 MHz XTAL is used as the system clock source and is divided down, then the clock source selected for the CAN interface must be the system clock (i.e., the divided 4 – 40 MHz XTAL) to keep the system clock not slower than the CAN interface clock.

5.5.2 FlexRay Clock Domains

The FlexRay block has two distinct software-controlled clock domains. One of the clock domains is always derived from the system clock. The source for the second clock domain can be the system clock or the 4 – 40 MHz XTAL output. The logic in the second clock domain controls the FlexRay interface pins. The CLK_SRC bit in the FlexRay CTRL register selects between the system clock and the oscillator clock

as the clock source for the second domain. Selecting the oscillator as the clock source ensures low jitter on the FlexRay bus.

NOTE

To prevent improper FlexRay behavior, the system clock or the FlexRay protocol engine clock source must be switched and stable before enabling the FlexRay module. After it is enabled, the FlexRay module can be disabled only by asserting $\overline{\text{RESET}}$.

If the oscillator clock source is selected for the FlexRay interface, then a divided down 4 – 40 MHz XTAL cannot be selected as the source for the system clock.

5.5.3 API / RTC Clock Domains

The clock source for the RTC can be selected as one of the following:

- 32 kHz_XTAL
- 4 – 40 MHz XTAL
- 128 kHz_IRC
- 16 MHz_ IRC

If the 32 kHz crystal is used (enabled), it is assumed that the crystal always remains the clock source.

The API/RTC includes a divide of the clock sources. The division of these clock sources can be configured to have a ÷512 and a ÷32. Both can be bypassed if required.

The 4 – 40 MHz XTAL and 16 MHz IRC can be optionally divided down by ÷1, ÷2, ÷4, ÷8, or ÷16 before being supplied to the API/RTC. This allows the required wake-up times and resolution to be met since the ÷32 may be too large for some users of the RTC to meet the desired resolution. The 128 kHz IRC can be divided by ÷1, ÷2, or ÷4. This allows the speed of the clock to be reduced to minimize power consumption in low power modes and to clock the RTC and 32 kHz.

For accurate RTC support, the 32 kHz_XTAL and 4 – 40 MHz XTAL clocks can be used. The 32 kHz_XTAL is normally used, but there are use cases to have the main 4 – 40 MHz XTAL crystal oscillator running in low power mode and clocking the RTC.

The maximum 4 – 40 MHz XTAL that can be run in low power mode is an 8 MHz crystal. The oscillator supports an automatically switched low power output buffer to minimize the current consumption when the 4 – 40 MHz XTAL is allowed to run. This keeps the overall low power mode current consumption down.

For low power API, the 128 kHz_IRC is used.

NOTE

To prevent improper real-time clock (RTC) behavior when switching the system clock source, or before the desired clock source has stabilized, the RTC must first be disabled by clearing the CRP_RTCC[CNTEN] = 0.

5.5.4 SWT Clock Domain

The clock source for the SWT is the 16 MHz_IRC.

5.5.5 Input/Output Processor (IOP) Clocking

The e200z0 IOP always runs at half the system frequency. If the system frequency source is the 16 MHz_IRC (e.g. after wake-up) the IOP is clocked at 8 MHz.

5.5.6 FEC Clocking

The Fast Ethernet Controller is not capable of running at the target system bus speed of the device. A permanent divide-by-two prescaler has been introduced into the clock tree for the FEC so that it always runs at half the speed of the system clock.

5.5.7 Media Local Bus (MLB) DIM Clocking

The MLB DIM always runs at half the system frequency.

Chapter 6
Clocks, Reset, and Power (CRP)

6.1 Introduction

The primary function of the clock, reset, and power (CRP) block is to maintain all of the control logic that requires power when other portions of the device are powered down in power-saving modes. The CRP manages entry into, operation during, and exit from power-saving modes.

The CRP consists of the input isolation block, the RTC/API, the wakeup and power status block, the clock and reset control block, low-power state machine, and bus interface unit. The input isolation block allows inputs from external blocks to be driven to known states when the logic driving the input is powered down. The RTC/API block implements a real-time counter and periodic interrupt. The wakeup and power status block implements the logic to select power mode operation and wakeup sources. The clock and reset control block implements miscellaneous logic related to PLL and oscillator operation, and reset gating for power-saving modes. The low-power state machine controls the transitions into and out of the power-saving modes. The bus interface unit allows for slave read/write register access from the device's core. There are also several miscellaneous integration functions included in the CRP that are discussed in detail in later sections of this chapter.

6.1.1 Block Diagram

A simplified block diagram of the CRP illustrates the functionality and interdependence of major blocks (see Figure 6-1).

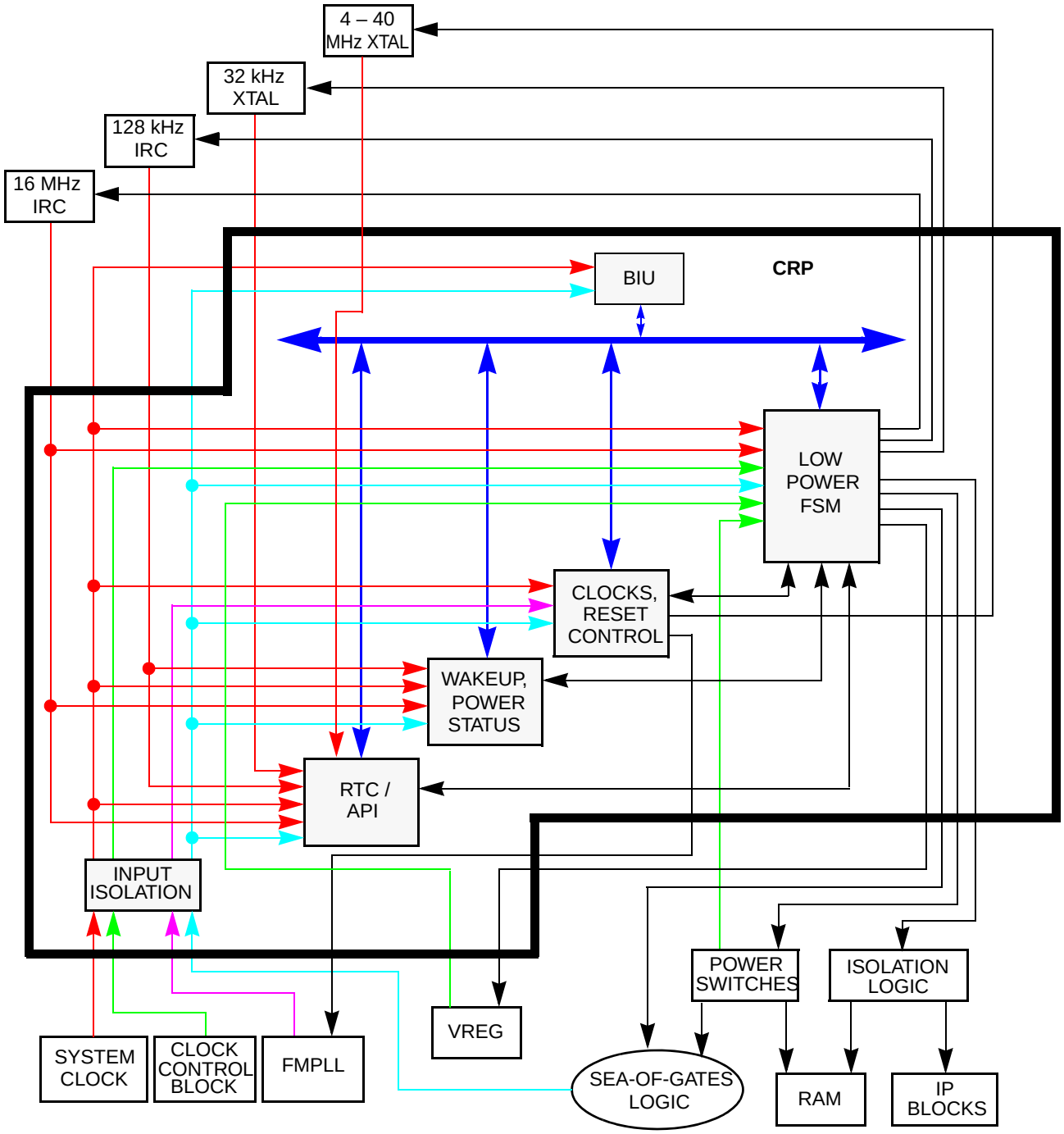


Figure 6-1. CRP Block Diagram

6.1.2 Features

The CRP has these major features:

- Real-time clock/autonomous periodic interrupt (RTC/API):
  - 32-bit counter

- Four selectable counter clock sources
  - 4 – 40 MHz XTAL with 1 to 16 divider
  - 32 kHz XTAL
  - 16 MHz_IRC with 1 to 16 divider stage
  - 128 kHz_IRC with 1 to 4 divider stage
- Optional divide-by-512 prescaler and optional divide-by-32 prescaler connected in series in the clock path feeding the 32-bit counter
- 32-bit counter supports times up to greater than 1.5 months with 1 ms resolution.
- 12-bit compare value to support interrupt intervals of 1 s up to greater than 1 hr with 1 s resolution
- RTC interrupt with interrupt enable.
- Counter runs in all modes of operation.
- RTC status and control register are reset only by POR
- RTC counter is reset when counter is disabled by software and by POR.
- Autonomous periodic interrupt support includes:
  - 10-bit compare value to support wakeup intervals of 1.0 ms to 1 s
  - Wakeup logic has separate enable to support changing compare value while RTC running
  - API interrupt with interrupt enable
  - Operates in all modes of operation
  - API compare value can be modified while RTC is running
- Optional interrupt for RTC match, API match, and RTC rollover.

- Low-power mode management:
  - Provides control of voltage regulator, LVI circuits, isolation enables, and power switches
  - FSM clock gates itself off when waiting for asynchronous wakeup signal for power savings
  - Four selections available for block sizes for RAM data retention (0, 32 KB, 64 KB, and 128 KB)
- Low-power wakeup:
  - Wakeup sources can be either the RTC, API, RTC rollover, or external pin
  - All wakeup sources can be enabled at any given time (first to occur generates wakeup)
  - 32 pin wakeup sources
  - Pin wakeup occurs on rising edge, falling edge, or both
  - Two clock-source inputs for pin wakeup to allow for lower power or faster wakeup
  - System level reset control to ensure clean recovery from sleep mode
- Miscellaneous:
  - All functional logic inputs isolated in low-power modes
  - All logic with multiple clock sources internally synchronized

6.1.3 Modes of Operation

There are two functional modes of operation for the CRP: normal operation and sleep mode.

In normal operation, all CRP registers can be read or written. The input isolation, low-power FSM, and wake-up logic are disabled. The voltage regulator, LVI, and power switch outputs are in the enabled state. The RTC/API and associated interrupts are optionally enabled.

In sleep mode, the bus interface is disabled and the input isolation is enabled. The RTC/API is enabled if enabled prior to entry into sleep. See Section 6.4.2, RTC Functional Description, for further details.

6.2 Memory Map and Registers

This section provides a detailed description of all CRP registers.

6.2.1 Module Memory Map

The CRP memory map is shown in Table 6-1. The address of each register is given as an offset to the CRP base address. Registers are listed in address order, identified by complete name and mnemonic, and lists the type of accesses allowed.

Table 6-1. CRP Memory Map

Offset from CRP_BASE (0xFFFE_C000)	Register	Access	Reset Value	Section/Page
0x0000	CRP_CLKSRC—Clock source register	R/W	0x0001_1F3F	6.2.2.1/6-5
0x0004–0x000F	Reserved			
0x0010	CRP_RTCC—RTC control register	R/W	0x0000_0000	6.2.2.2/6-6
0x0014	CRP_RTSC—RTC status register	R	0x0000_0000	6.2.2.3/6-8
0x0018	CRP_RTCCNT—RTC counter register	R	0x0000_0000	6.2.2.4/6-9
0x001C–0x003F	Reserved			
0x0040	CRP_PWKENH—Pin wakeup enable high register	R/W	0x0000_0000	6.2.2.5/6-9
0x0044	CRP_PWKENL—Pin wakeup enable low register	R/W	0x0000_0000	6.2.2.5/6-9
0x0048	CRP_PWKSRCIE—Pin wakeup source interrupt enable register	R/W	0x0000_0000	6.2.2.6/6-11
0x004C	CRP_PWKSRCF—Pin wakeup source flag register	R	0x0000_0000	6.2.2.7/6-11
0x0050	CRP_Z6VEC—Z6 reset vector register	R/W	0xFFFF_0001	6.2.2.8/6-12
0x0054	CRP_Z0VEC—Z0 reset vector register	R/W	0xFFFF_FFFE	6.2.2.9/6-12
0x0058	CRP_RECPTR—Recovery pointer register	R/W	0xFFFF_FFFC	6.2.2.10/6-13
0x005C–0x005F	Reserved			
0x0060	CRP_PSCR—Power status and control register	R/W	0x0000_0000	6.2.2.11/6-14
0x0064–0x006F	Reserved			

Table 6-1. CRP Memory Map (continued)

Offset from CRP_BASE (0xFFFE_C000)	Register	Access	Reset Value	Section/Page
0x0070	CRP_SOCSC—SoC status and control register	R/W	0x4000_0000	6.2.2.12/6-15
0x0074–0x03FF	Reserved			

6.2.2 Register Descriptions

This section lists the CRP registers in address order and describes the registers and their bit fields.

6.2.2.1 Clock Source Register (CRP_CLKSRC)

The CRP_CLKSRC contains:

- Enable bits for the 32 kHz OSC, 128 kHz IRC, and 4 – 40 MHz OSC
- Low power configuration for the 4 – 40 MHz OSC
- The trim values for the 16 MHz IRC and 128 kHz IRC

Offset: CRP_BASE + 0x0000 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	IRC TRIM EN	0	0	0	0	PREDIV			0	0	0	0	EN128K IRC	EN32 KOSC	ENLP OSC	EN40M OSC
W																
Reset[1]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	TRIM128IRC					0	0	TRIM16IRC					
W																
Reset[1]	0	0	0	1	1	1	1	1	0	0	1	1	1	1	1	1

[1] These bits are only reset by power-on, VDD12 LVI, VDD33 LVI, VDDSYN LVI, VDD5 Low LVI, and VDD5 LVI.

Figure 6-2. Clock Source Register (CRP_CLKSRC)

Table 6-2. CRP_CLKSRC Field Descriptions

Field	Description
IRCTRIMEN	IRC Trim Enable. The IRCTRIMEN bit enable write access to TRIM128IRC and TRIM16IRC. 0 IRC trim bit writes disabled. 1 IRC trim bit writes enabled.
PREDIV	RTC Clock Pre-divider. The PREDIV bits control the pre-divider for the RTC clock source. Divide clock sources are the 32 kHz OSC, 128 kHz IRC, 16 MHz IRC, or the 4 – 40 MHz OSC. See the CLKSEL bitfield in Table 6-3.The pre-divider is in addition to any other divide selects internal to the RTC logic. 000 Divide by 1 001 Divide by 2 010 Divide by 4 011 Divide by 8 100 Divide by 16 Other Reserved

Table 6-2. CRP_CLKSRC Field Descriptions (continued)

Field	Description
EN128KIRC	Enable 128 kHz IRC Oscillator. The EN128KIRCbit enables the 128 kHz IRC oscillator. 0 128 kHz IRC disabled. 1 128 kHz IRC enabled. **Note:** After enabling the128 kHz IRC, software needs to wait the required crystal startup/stabilization time before making use of this oscillator.
EN32KOSC	Enable 32 kHz Oscillator. The EN32KOSC bit enables the 32 kHz XTAL oscillator. 0 32 kHz OSC disabled. 1 32 kHz OSC enabled. **Note:** After enabling the 32 kHz OSC, software needs to wait the required crystal startup/stabilization time before making use of this oscillator.
ENLPOSC	Enable Low Power External Oscillator. The ENLPOSC bit controls how the 4 – 40 MHz OSC behaves during sleep if EN40MOSC is high. If EN40MOSC is low, then ENLPOSC has no effect. 0 4 – 40 MHz OSC clock disabled to save power, but connection to the external crystal is still active thus supports faster recovery time for availability of 4 – 40 MHz OSC after sleep recovery. Supports full 4 – 40 MHz range of external crystals. 1 4 – 40 MHz OSC clock active and may be used as clock source for RTC/API. The external crystal frequency is limited to ≤ 8 MHz.
EN40MOSC	Enable 4 – 40 MHz Oscillator Enable. The EN40MOSC bit enables the 4 – 40 MHz OSC external oscillator for external crystal. 0 4 – 40 MHz OSC disabled. 1 4 – 40 MHz OSC enabled. **Note:** During sleep mode with EN40MOSC = 1, the 4 – 40 MHz OSC still can actively drive an external crystal and can be used to clock the RTC/API if the crystal frequency is ≤ 8 MHz.
TRIM128IRC	Trim Value for 128 kHz IRC. The TRIM128IRC bits are a 2's complement trimming method, so the trimming code increases from –16 to +15. The default trimming code is 0b11111 (–1, nearly in the middle of –16 and +15). As the code increases/decreases the frequency reduces/increases. TRIM128IRC can only be updated if IRCTRIMEN is enabled.
TRIMIRC	Trim Value for 16 MHz IRC. The TRIM16IRC bits are a 2's complement trimming method, so the trimming code increases from –32 to +31. The default trimming code is 0b111111 (–1, nearly in the middle of –32 and +31). As the code increases/decreases the frequency reduces/increases. TRIM16IRC can only be updated if IRCTRIMEN is enabled.

6.2.2.2 RTC Control Register (CRP_RTCC)

The CRP_RTCC register contains:

- RTC counter enable
- RTC interrupt enable
- RTC clock source select
- RTC compare value
- API enable
- API interrupt enable
- API compare value

Offset: CRP_BASE + 0x0010 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	CNTEN	RTCIE	FRZEN	ROVREN	RTCVAL											
Reset[1]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	APIEN	APIIE	CLKSEL		DIV512EN	DIV32EN	APIVAL									
Reset[1]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

1 These bits are only reset by power-on: VDD15 LVI, VDD33 LVI, and VDDSYN LVI, VDD5 low LVI, and VDD5 LVI.

Figure 6-3. RTC Control Register (CRP_RTCC)

Table 6-3. CRP_RTCC Field Descriptions

Field	Description
CNTEN	Counter Enable. The CNTEN bit enable the RTC counter. Making CNTEN bit 1'b0 has the effect of asynchronously resetting (synchronous reset negation) all the RTC and API logic. This allows for the RTC configuration and clock source selection to be updated without causing synchronization issues. 0 Counter disabled. 1 Counter enabled.
RTCIE	RTC Interrupt Enable. The RTCIE bit enables interrupts requests to the system if RTCF is asserted. 0 RTC interrupts disabled. 1 RTC interrupts enabled.
FRZEN	Freeze Enable Bit. The counter freezes on entering the debug mode (as the ipg_debug is detected active) on the last valid count value if the FRZEN bit is set. After coming out of the debug mode, the counter starts from the frozen value. 0 Counter does not freeze in debug mode. 1 Counter freezes in debug mode.
ROVREN	Counter Roll Over Wakeup/Interrupt Enable. The ROVREN bit enables wakeup and interrupt requests when the RTC has rolled over from 0xFFFF_FFFF to 0x0000_0000. The RTCIE bit must also be set in order to generate an interrupt from a counter rollover. 0) RTC rollover wakeup/interrupt disabled 1) RTC rollover wakeup/interrupt enabled.
RTCVAL	RTC Compare Value. The RTCVAL bits are compared to bits 21–10 of the RTC counter. If they match, RTCF is set. **Note:** RTCVAL must be non-zero for a match to occur.
APIEN	Autonomous Periodic Interrupt Enable. The APIEN bit enables the autonomous periodic interrupt function. 0 API disabled. 1 API enabled.
APIIE	API Interrupt Enable. The APIIE bit enables interrupts requests to the system if APIF is asserted. 1 API interrupts enabled. 0 API interrupts disabled.

Table 6-3. CRP_RTCC Field Descriptions (continued)

Field	Description
CLKSEL	Clock Select. The CLKSEL bits select the clock source for the RTC. CLKSEL should only be updated when CNTEN is 0. The user should ensure that oscillator is enabled before selecting it as a clock source for RTC. 00 32 kHz OSC 01 128 kHz IRC 10 16 MHz IRC 11 4 – 40 MHz OSC
DIV512EN	Divide by 512 enable. The DIV512EN bit enables the 512 clock divider. DIV512EN should only be updated when CNTEN is 0. 0 Divide by 512 is disabled. 1 Divide by 512 is enabled
DIV32EN	Divide by 32 enable. The DIV32EN bit enables the 32 clock divider. DIV32EN should only be updated when CNTEN is 0. 0 Divide by 32is disabled. 1 Divide by 32 is enabled
APIVAL	API Compare Value. The APIVAL bits are compared to an offset value based on bits 22–31 of the RTC counter. If they match, a wakeup/interrupt request is asserted. APIVAL can be updated only when APIEN = 0 or when the API function is undefined. **Note:** API functionality starts only when APIVAL is non-zero. The first API interrupt takes two more cycles because of synchronization of APIVAL to the RTC clock. After that, interrupts are periodic in nature. The compare value is API + 1 and the minimum supported value is 4, because of synchronization issues.

6.2.2.3 RTC Status Register (CRP_RTSC)

The CRP_RTSC register contains:

- RTC interrupt flag
- API interrupt flag
- ROLLOVR Flag

Offset: CRP_BASE + 0x0014 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	RTCF	0	0	0	0	0	0	0	0	0	0	0	0	0
W			w1c													
Reset[1]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	APIF	0	0	ROVRF	0	0	0	0	0	0	0	0	0	0
W			w1c			w1c										
Reset[1]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

1 These bits are only reset by power-on: VDD15 LVI, VDD33 LVI, and VDDSYN LVI, VDD5 low LVI, and VDD5 LVI.

Figure 6-4. RTC Status Register (CRP_RTSC)

Table 6-4. CRP_RTSC Field Descriptions

Field	Description
RTCF	RTC Interrupt Flag. The RTCF bit indicates that the RTC counter has reached the counter value matching RTCVAL. RTCF is cleared by writing a 1 to RTCF. Writing a 0 to RTCF has no effect. Note that the RTCF bit must be cleared before entering sleep mode, if the RTC is to be used as the wakeup source. 0 No RTC interrupt. 1 RTC interrupt.
APIF	API Interrupt Flag. The APIF bit indicates that the RTC counter has reached the counter value matching API offset value. APIF is cleared by writing a 1 to APIF. Writing a 0 to APIF has no effect. Note that the APIF bit must be cleared before entering SLEEP mode, if the API is to be used as the wakeup source. 0 No API interrupt. 1 API interrupt.
ROVRF	Counter Roll Over Interrupt Flag. The ROVRF bit indicates that the RTC has rolled over from 0xFFFF_FFFF to 0x0000_0000. ROVRF is cleared by writing a 1 to ROVRF. Writing a 0 to ROVRF has no effect. Note that the ROVRF bit must be cleared before entering sleep mode, if the RTC rollover is to be used as the wakeup source. 0) RTC has not rolled over. 1) RTC has rolled over.

6.2.2.4 RTC Counter Register (CRP_RTCCNT)

The CRP_RTCCNT register contains the RTC counter value.

Offset: CRP_BASE + 0x0018 Access: User read only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	RTCCNT																															
W																																
Reset[1]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

1 These bits are only reset by power-on: VDD15 LVI, VDD33 LVI, and VDDSYN LVI, VDD5 low LVI, and VDD5 LVI.

Figure 6-5. RTC Counter Register (CRP_RTCCNT)

Table 6-5. CRP_RTCCNT Field Descriptions

Field	Description
RTCCNT	RTC Counter Value. Due to the clock synchronization, the RTCCNT value may actually represent a previous counter value

6.2.2.5 Pin Wakeup Enable Registers (CRP_PWKENH/L)

The CRP_PWKENH and CRP_PWKENL registers enable the pin wakeup sources and select the trigger edge for the wakeup inputs.

Offset: CRP_BASE + 0x0040 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	PWK31		PWK30		PWK29		PWK28		PWK27		PWK26		PWK25		PWK24	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	PWK23		PWK22		PWK21		PWK20		PWK19		PWK18		PWK17		PWK16	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 6-6. Pin Wakeup Enable High Register (CRP_PWKENH)

Offset: CRP_BASE + 0x0044 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	PWK15		PWK14		PWK13		PWK12		PWK11		PWK10		PWK9		PWK8	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	PWK7		PWK6		PWK5		PWK4		PWK3		PWK2		PWK1		PWK0	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 6-7. Pin Wakeup Enable Low Register (CRP_PWKENL)

Table 6-6. CRP_PWKENH/L Field Descriptions

Field	Description
PWK*n*	Pin Wakeup Enable and Select. The PWK*n* bits enable the external pin wakeup sources and define which edge transition is used for the wakeup. 00 External pin wakeup source disabled. 01 Positive edge of selected external pin triggers the wakeup request. 10 Negative edge of selected external pin triggers the wakeup request. 11 Positive or negative edge of selected external pin triggers the wakeup request.

Table 6-7. Wakeup Source Number vs. Pin

CRP_PWKENL				CRP_PWKENH			
PWK*n*	**Pin**	**PWK*n***	**Pin**	**PWK*n***	**Pin**	**PWK*n***	**Pin**
0	PB4	**8**	PD9	**16**	PE9	**24**	PJ9
1	PB5	**9**	PD11	**17**	PE11	**25**	PJ10
2	PB6	**10**	PD13	**18**	PE13	**26**	PJ11
3	PB7	**11**	PD15	**19**	PF3	**27**	PJ12
4	PD1	**12**	PE1	**20**	PF7	**28**	PJ13

Table 6-7. Wakeup Source Number vs. Pin (continued)

CRP_PWKENL				CRP_PWKENH			
PWK*n*	**Pin**	**PWK*n***	**Pin**	**PWK*n***	**Pin**	**PWK*n***	**Pin**
5	PD3	**13**	PE3	**21**	PF11	**29**	PJ14
6	PD5	**14**	PE5	**22**	PF15	**30**	PK3
7	PD7	**15**	PE7	**23**	PJ8	**31**	PK6

NOTE

Program any pins that are to be used as wakeup sources as inputs in the associated SIU_PCR*n* register prior to entering a low-power mode.

6.2.2.6 Pin Wakeup Source Interrupt Enable Register (CRP_PWKSRCIE)

The CRP_PWKSRCIE register enables interrupt requests individually for each of the pin wakeup sources.

Offset: CRP_BASE + 0x0048 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	PWKSRCIE31-0																															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 6-8. Pin Wakeup Source Interrupt Enable Register (CRP_PWKSRCIE)

Table 6-8. CRP_PWKSRCIE Field Descriptions

Field	Description
PWKSRCIE*n*	Pin Wakeup Source Interrupt Enables. The PWKSRIE*n* bits enable interrupt requests to the system if the corresponding PWK*n* bit in CRP_PWKENH/L is asserted. 0 Wakeup source interrupt disabled. 1 Wakeup source interrupt enabled.

6.2.2.7 Pin Wakeup Source Flag Register (CRP_PWKSRCF)

The CRP_PWKSRCF register indicates external pin wakeup source events.

Offset: CRP_BASE + 0x004C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	PWKSRCF[31:0]																															
W	w1c																															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 6-9. Pin Wakeup Source Flag Register (CRP_PWKSRCF)

Table 6-9. CRP_PWKSRCF Field Descriptions

Field	Description
PWKSRCF*n*	Pin Wakeup Source Flag. The PWKSRCF bits indicate which external pin wakeup source event caused the wakeup. More than one external wakeup source can be asserted at a time if the wakeup events happened simultaneously. A write of 1 clears the interrupt flag. A write of 0 has no effect. 0 PWKSRCF*n* did not cause the last wakeup. 1 PWKSRCF*n* caused the last wakeup.

6.2.2.8 Z6 Reset Vector Register (CRP_Z6VEC)

The CRP_Z6VEC register contains:

- Recovery vector for the Z6 core
- Reset for the Z6 core
- VLE select for the Z6 core

Offset: CRP_BASE + 0x0050 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	Z6VEC																				0	0	0	0	0	0	0	0	0	0	Z6RST	VLE
W																																
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	1

Figure 6-10. Z6 Reset Vector Register (CRP_Z6VEC)

Table 6-10. CRP_Z6VEC Field Descriptions

Field	Description
Z6VEC	The user needs to change this value to point to a different memory location for system reinitialization when exiting low-power sleep mode. The program counter value for the Z6 after Sleep mode recovery is {Z6VEC, 0xFFC}, which aligns to a 4 KB address boundary offset by 0xFFC. Note that the default Z6 MMU configuration is a 4 KB memory space aligned to the 4 KB address boundary defined by Z6VEC. Thus the reinitialization code needs to access within the 4 KB memory space until the MMU is reconfigured.
Z6RST	Controls the assertion of RESET to the Z6 core. Writes to this bit cause the Z6 to immediately enter/exit reset. Reads of this bit indicate if the core is being held in reset. 0 Z6 not in reset. 1 Z6 in reset.
VLE	VLE Select. The VLE bit selects whether the Z6 recovers into VLE or Book_E mode. 0 Z6 recovers into Book_E mode. 1 Z6 recovers into VLE mode.

NOTE

The user may attempt to set both the CRP_Z6VEC[Z6RST] and CRP_Z0VEC[Z0RST] bits to 1, but if one of these bits is already set to a value of 1, the write to the other bit is blocked.

6.2.2.9 Z0 Reset Vector Register (CRP_Z0VEC)

The CRP_Z0VEC register contains:

- Recovery vector for the Z0 core
- Reset for the Z0 core

Offset: CRP_BASE + 0x0054 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	Z0VEC																														Z0RST	0
W																																
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0

Figure 6-11. Z0 Reset Vector Register (CRP_Z0VEC)

Table 6-11. CRP_Z0VEC Field Descriptions

Field	Description
Z0VEC	Z0 Recovery Vector. The Z0VEC value determines the initial program counter for the Z0 when exiting reset. On reset, the value contained in the register defaults to 0xFFFF_FFFE, and the Z0 is held in reset. Change this value to point to a different memory location for Z0 specific initialization when negating the Z0RST bit or recovering from Sleep mode.
Z0RST	Controls the assertion of RESET to the Z0 core. Writes to this bit cause the Z0 to immediately enter/exit reset. Reads of this bit indicate if the core is being held in reset. 0 Z0 not in reset. 1 Z0 in reset.

NOTE

The user may attempt to set the CRP_Z6VEC[Z6RST] and CRP_Z0VEC[Z0RST] bits to 1, but if one of these bits is already set to a value of 1, the write to the other bit is blocked.

6.2.2.10 Reset Recovery Pointer Register (CRP_RECPTR)

The CRP_RECPTR register contains:

- Recovery pointer
- Fast recovery enable

Offset: CRP_BASE + 0x0058 Access: User read only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	RECPTR																														FAST REC	0
W																																
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0

Figure 6-12. Reset Recovery Pointer (CRP_RECPTR)

Table 6-12. CRP_RECPTR Field Descriptions

Field	Description
RECPTR	Recovery Pointer. The RECPTR value is a generic 30-bit register available to the user application which retains a value during all low-power modes. This register may be used by the user software to indicate where in RAM a recovery routine exists.
FASTREC	Fast Reset Recovery. Allows the reset sequence generated at the exit of a sleep mode to be shortened to 16 clocks. This bit may be used when the CRP_Z6VEC or CRP_Z0VEC register of the core(s) executing code after a sleep mode points to a memory other than the flash. This allows code to be executed from those other memories while the flash completes its internal initialization. 0 Reset occurs for 1000 clocks. 1 Reset occurs for 16 clocks.

6.2.2.11 Power Status and Control Register (CRP_PSCR)

The power status and control register (CRP_PSCR) contains:

- Wakeup mode and source flags
- Sleep mode enable
- Sleep RAM retention select

Offset: CRP_BASE + 0x0060 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	SLEEPF	0	0	0	0	0	0	0	0	0	0	0	0	RTCOVRWKF	RTCWKF	APIWKF
W	w1c													w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	SLEEP	0	0	0	0	RAMSEL			0	0	0	0	WKCLKSEL	RTCOVRWKEN	RTCWKEN	APIWKEN
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 6-13. Power Status and Control Register (CRP_PSCR)

Table 6-13. CRP_PSCR Field Descriptions

Field	Description
SLEEPF	SLEEP Flag. The SLEEPF bit indicates whether recovery from the last low-power modes was sleep. While SLEEPF is set, the pads remain in a safe state after Sleep mode recovery ad clearing SLEEPF will return the pads to normal operation. A write of 1 clears this status flag and a write of 0 has no effect. 0 Low-power sleep mode was not entered. 1 Low-power sleep mode entered.
RTCOVRWKF	RTC Counter Rollover Wakeup Flag. The RTCOVRWKF bit indicates that a RTC counter rollover was the wakeup source. A write of 1 clears the interrupt flag and a write of 0 has no effect. 0 The RTC counter did not cause the last wakeup. 1 The RTC counter caused the last wakeup.

Table 6-13. CRP_PSCR Field Descriptions (continued)

Field	Description
RTCWKF	RTC Wakeup Flag. The RTCWKF bit indicates that the RTC match was the wakeup source. A write of 1 clears the interrupt flag and a write of 0 has no effect. 0 The RTC did not cause the last wakeup. 1 The RTC caused the last wakeup.
APIWKF	API Wakeup Flag. The APIWKF bit indicates the API was the wakeup source. A write of 1 clears the interrupt flag and a write of 0 has no effect. 0 The API did not cause the last wakeup. 1 The API caused the last wakeup.
SLEEP	SLEEP Request. The SLEEP bit indicates a request to enter the sleep low-power mode. This bit is cleared automatically when exiting from SLEEP. 0 No request to enter the sleep low-power mode. 1 Request to enter the sleep low-power mode.
RAMSEL	RAM Selects. The RAMSEL bits select which ram configuration retains power during the sleep mode. 000 All RAMs powered down 001 32 KB RAM retains power (0x4000_0000 – 0x4000_7FFF). 010 64 KB RAM retains power (0x4000_0000 – 0x4000_FFFF). 011 128 KB RAM retains power (0x4000_0000 – 0x4001_FFFF). Other reserved.
WKCLKSEL	Wakeup Clock Select. The WKCLKSEL bit selects the clock source used for the wakeup logic synchronizer and edge detect. WKCLKSEL can be switched only when all wakeup sources are disabled. 0 Clock source for wakeup logic is the 128 kHz IRC. 1 Clock source for wakeup logic is the 16 MHz IRC. **Note:** The 128 kHz IRC is not automatically enabled if selected; therefore, it must be enabled before it is selected for use. **Note:** When using the 128 kHz IRC to wake up from SLEEP, the application software must wait at least one 128 kHz clock cycle after entering SLEEP before waking up. **Note:** The wakeup flag cannot be cleared until at least three 128 kHz cycles after it has been set.
RTCOVRWKEN	RTC Rollover Wakeup Enable. The RTCOVRWKEN bit enables a rollover of the RTC counter to be a wakeup source for exit from low-power modes. 0 RTC rollover does not generate a wakeup request from low-power mode. 1 RTC rollover generates a wakeup request from low-power modes.
RTCWKEN	RTC Wakeup Enable. The RTCWKEN bit enables the RTC to be a wakeup source for exit from low-power modes. 0 RTC not enabled as a wakeup source. 1 RTC enabled as a wakeup source.
APIWKEN	API Wakeup Enable. The APIWKEN bit enables the API to be a wakeup source for exit from low-power modes. 0 The API does not generate a wakeup request from low-power mode. 1 The API generates a wakeup request from low-power modes.

6.2.2.12 SoC Status and Control Register (CRP_SOCSC)

The CRP_SOCSC register contains:

- LVI interrupt flags
- LVI interrupt enables

- LVI reset enables
- LVI lock bit

Offset: CRP_BASE + 0x0070 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	LVI5 LOCK	LVI5 RE	0	0	0	0	0	0	0	LVI5H IE	LVI5N IE	LVI5 IE	0	0	FRIE	FDIS
W																
Reset	0[1]	1[1]	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	LVI5HIF	LVI5NF	LVI5F	0	0	FRF	FRDY
W										w1c	w1c	w1c			w1c	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

[1] These bits are only reset by power on, VDD15 LVI, VDD33 LVI, VDDSYN LVI, and VDD5 Low LVI.

Figure 6-14. SoC Status and Control Register (CRP_SOCSC)

Table 6-14. CRP_SOCSC Field Descriptions

Field	Description
LVI5LOCK	LVI5 Lock. The LVI5LOCK bit disables writes to the LVI5RE register bit. After it is set, this bit remains set until the next POR. 0 LVI5RE writeable. 1 LVI5RE not writeable.
LVI5RE	LVI5 Reset Enable. The LVI5RE bit enables the reset function of the LVI5. 0 LVI5 does not generate a reset when LVI5F is set. 1 LVI5 generates a reset when the LVI5F is set. **Note:** When the VRCSEL pin is low, the 5V LVI logic is disabled and this bit has no effect.
LVI5HIE	LVI5 High Interrupt Enable. TheLVI5HIE bit enables interrupts requests to the system if LVI5HF is asserted. 0 LVI5H interrupts disabled. 1 LVI5H interrupts enabled. **Note:** When the VRCSEL pin is low, the 5V LVI logic is disabled and this bit has no effect.
LVI5NIE	LVI5N Interrupt Enable. TheLVI5NIE bit enables interrupts requests to the system if LVI5NF is asserted. 0 LVI5N interrupts disabled. 1 LVI5N interrupts enabled. **Note:** When the VRCSEL pin is low, the 5V LVI logic is disabled and this bit has no effect.
LVI5IE	LVI5 Interrupt Enable. TheLVI5IE bit enables interrupts requests to the system if LVI5F is asserted. 0 LVI5 interrupts disabled. 1 LVI5 interrupts enabled. **Note:** When the VRCSEL pin is low, the 5V LVI logic is disabled and this bit has no effect.
FRIE	Flash Ready Interrupt Enable. The FRIE bit enables an interrupt that is generated based on the value of FRF. This notifies the user that the Flash is ready and available for read/write operations. 0 FRF interrupts disabled. 1 FRF interrupts enabled.
FDIS	Flash Disable. The FDIS bit places the flash into a disabled low power state. See Section 12.4.4, Flash Sleep Mode, for more information. 0 Flash enabled. 1 Flash disabled.

Table 6-14. CRP_SOCSC Field Descriptions (continued)

Field	Description
LVI5HF	LVI 5V High Interrupt Flag. The LVI5HF bit indicates that the LVI5H LVI circuit has detected that the 5V supply is below the trip limit. LVI5HF is cleared by writing a 1 to LVI5HF. Writing a 0 to LVI5HF has no effect. 0 No LVI5H interrupt. 1 LVI5H interrupt. **Note:** The LVI5H LVI circuit is disabled when VRCSEL is low and thus LVI5HF remains cleared.
LVI5NF	LVIN 5V Interrupt Flag. The LVI5NF bit indicates that the LVI5 LVI circuit has detected that the 5 V supply is above the defined trip limit. LVI5NF is cleared by writing a 1 to LVI5NF. Writing a 0 to LVI5NF has no effect. **Note:** If the supply remains above the defined trip limit, the LVI5NF flag is immediately re-set after the clear sequence. 0 No LVI5 negation interrupt. 1 LVI5 negation interrupt. **Note:** The LVI5 LVI circuit is disabled when VRCSEL is low and thus LVI5NF remains set.
LVI5F	LVI 5V Interrupt Flag. The LVI5F bit indicates that the LVI5 LVI circuit has detected that the 5 V supply is below the defined trip limit. LVI5F is cleared by writing a 1 to LVI5F. Writing a 0 to LVI5F has no effect. **Note:** If the supply remains below the defined trip limit, the LVI5F flag is immediately re-set after the clear sequence. 0 No LVI5 assertion interrupt. 1 LVI5 assertion interrupt. **Note:** The LVI5 LVI circuit is disabled when VRCSEL is low and thus LVI5F remains cleared.
FRF	Flash Ready Flag. The FRF bit is set when the Flash becomes available for read/write operations after recovery from Sleep or reset. FRF is cleared by writing a 1 to FRF. Writing a 0 to FRF has no effect. exiting low Power mode. It is used to notify the user software that Flash operation may begin. 0 Flash not ready. 1 Flash ready.
FRDY	Flash Ready. The FRDY bit is a real time indication of whether the Flash is ready for read/write operations after recovery from Sleep or reset. 0 Flash not ready. 1 Flash ready.

6.3 Functional Description

6.3.1 Low-Power Mode

The CRP supports a low power mode of operation, Sleep. During Sleep, the CRP logic remains powered and is not reset. The standard cell logic is powered down in Sleep mode. In order to achieve the functional requirements of this low power mode, the CRP provides the following functionality: control of the on-chip voltage regulator, LVI circuits, and power gates; wakeup monitoring on external pins or internal RTC/API; external reset pin monitoring to allow user to abort the low power mode; system recovery on wakeup; and support for JTAG and Nexus debug capability. The following sections discuss in detail the entry sequence, the operation, and the exit sequence for the low power Sleep mode.

6.3.2 Wake-Up Lines

The wake-up lines are implemented as described in Chapter 8, System Integration Unit (SIU), and detailed in Section 6.2.2.5, Pin Wakeup Enable Registers (CRP_PWKENH/L). These wake-up signals generate an

interrupt when the device wakes up. Each external wake-up has individual wakeup flag and interrupt enable and are grouped together into one interrupt vector. Refer to Chapter 3, Signal Description, for details on the allocation of pins to the Wake-up lines.

In order to minimize spurious wake-up as a result of noise, fixed duration input filters are applied to every wake-up pin.These filters are based on either the 128 kHz or 16 MHz clock sources and use 2 clock cycles to synchronize the input wake-up signal.

6.3.3 Low-Power Mode Entry

The sequence to enter the low-power sleep mode is for the user to disable the DMA, MLB, FEC, and FlexRay masters. Then halt all modules via the SIU_HLT registers. The system clock source should be set to the 16 MHz IRC prior to disabling the PLL or powering down the 4 – 40 MHz XTAL. The PLL should then be disabled since it does not clock any logic in sleep mode.

The main external oscillator (4 – 40 MHz XTAL) can be optionally powered down in sleep mode by clearing the EN40MOSC bit in the CRP_CLKSRC register. If EN40MOSC is enabled during Sleep, and EN40MOSC is 0, then the 4 – 40 MHz OSC can not be used as a clock source during sleep, but is still actively driving the external crystal which may support the full 4 – 40 MHz frequency range. If EN40MOSC is enabled during Sleep, and EN40MOSC is 1, then the 4 – 40 MHz XTAL may be used as a clock source for the RTC/API during sleep, but the external crystal frequency is limited to less than or equal to 8 MHz.

If the 4 – 40 MHz XTAL is powered down for sleep mode, the crystal oscillator must restart on the exit from sleep mode. If the 4 – 40 MHz XTAL powered down option is chosen, the user must be sure to first disable any logic that is being clocked directly by the 4 – 40 MHz XTAL to prevent glitches.

All program and erase operations on the flash array need to be completed before entering sleep mode.

Prior to entry into sleep mode, the ADC halt bit must be set or the ADC must be disabled. When exiting sleep mode, the required recovery time must elapse before the ADC can be enabled or the ADC halt bit is cleared. The recovery time allows the ADC circuits to stabilize. See the *PXN20 Microcontroller Data Sheet* for recovery times.

Sleep mode selection is done by setting the SLEEP bit in the CRP_PSCR register. With this bit set, each active core should individually execute the WAIT instruction to enter sleep mode. If only one core is active, and one is held in reset by the user, then executing the WAIT instruction on the active core initiates entry into the low-power mode. At this point, the CRP takes over operation of the device until a wakeup event occurs.

6.3.3.1 CRP Clock Selection

In sleep mode, the CRP control logic is clocked by the 16 MHz IRC. The RTC/API can be clocked by the 128 kHz_IRC, the 32 kHz_XTAL, the 16 MHz IRC, or the 4 – 40 MHz XTAL (restricted to less than or equal to 8 MHz). The pin wakeup logic can be clocked by either the 128 kHz IRC or the 16 MHz IRC. These clock source selections must be made prior to executing both WAIT instructions to the cores.

6.3.3.2 Sleep Mode RAM Retention

The RAMSEL bits in the CRP_PSCR register determine the amount of RAM that remains powered in sleep mode. This selection must be made prior to executing the WAIT instructions to the cores with the CRP_PSCR[SLEEP] bit set.

6.3.4 Low-Power Operation

After the WAIT instructions have been executed with the SLEEP bit set, and the cores have cleanly halted, the clock control block signals the CRP to enter the selected low-power mode.

At this point, the CRP has complete control of the device. Figure 6-15 shows the sequence to transition from RUN mode to SLEEP. Figure 6-16 and Figure 6-17 give the transition diagram for going from RUN mode to sleep, and then back to RUN mode.

The pads are put into a safe state during entry into Sleep mode. While is a safe state, the pad output buffers, input buffers, and pull devices are disabled. The RESET pad retains its function. The input buffers for the 32 wakeup pins retain the same state as previous to Sleep mode entry. The TDO pin is still active if debug is enabled before Sleep mode entry.

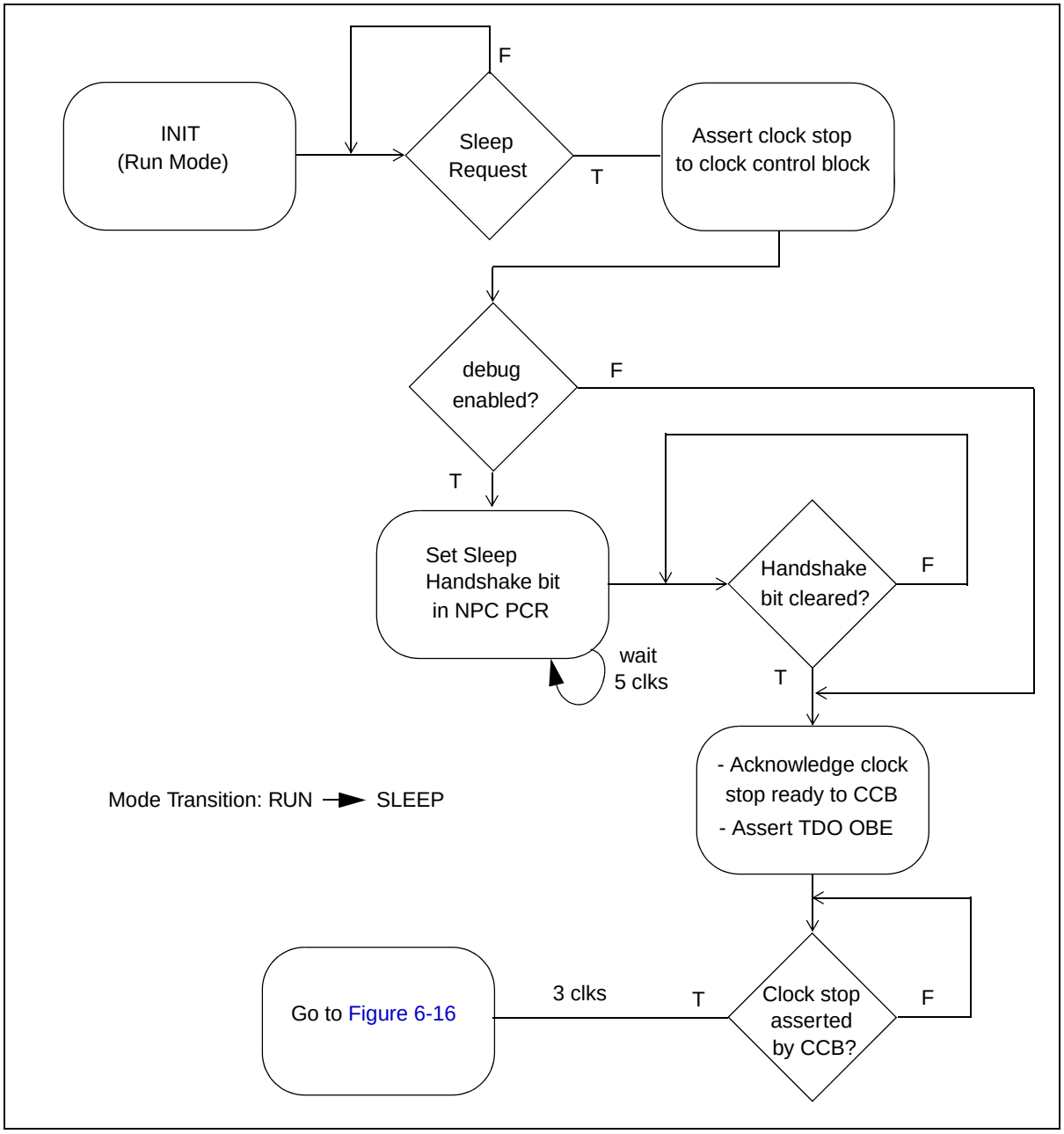


Figure 6-15. SLEEP Mode Entry Diagram

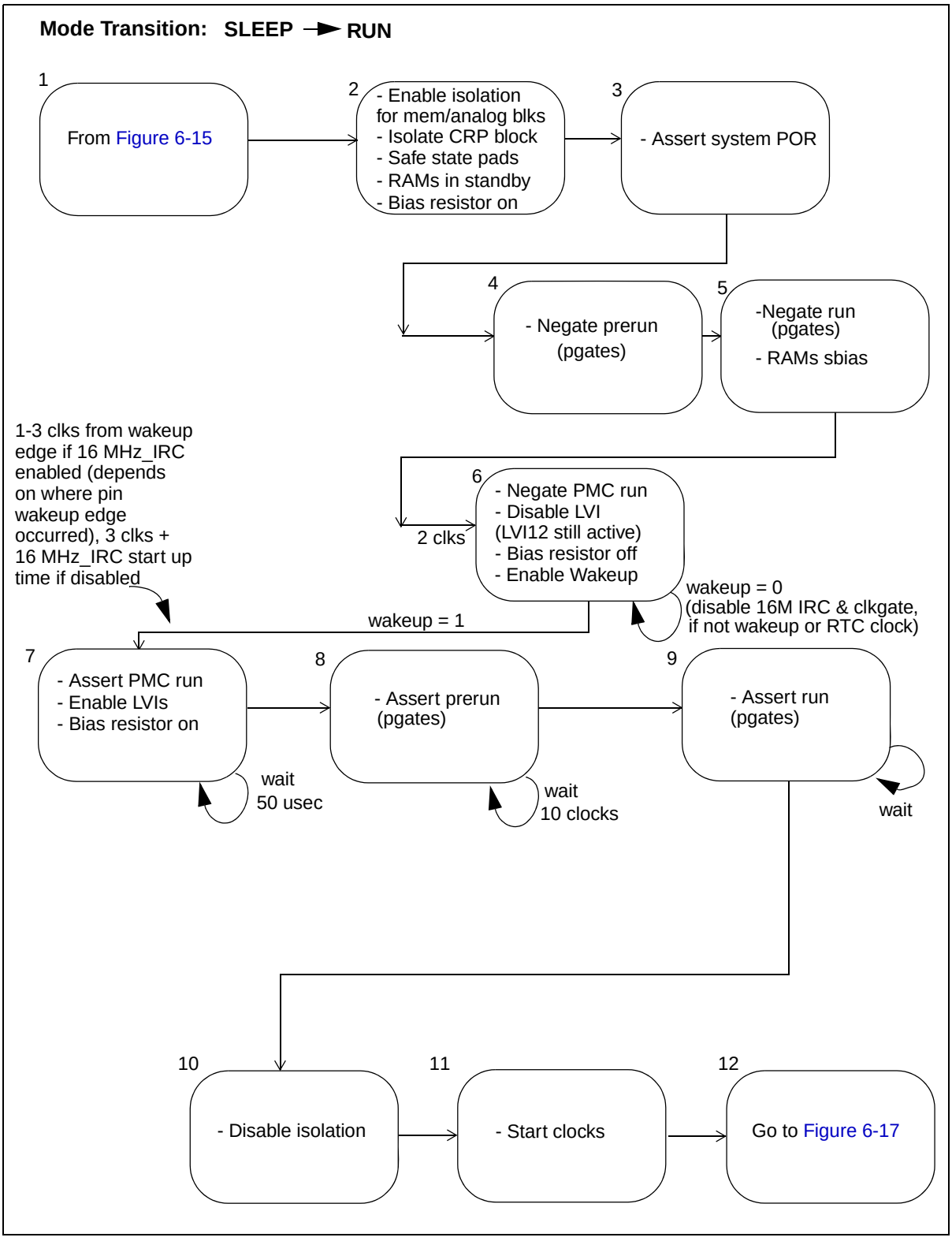


Figure 6-16. SLEEP Mode Transition Diagram (Part 1)

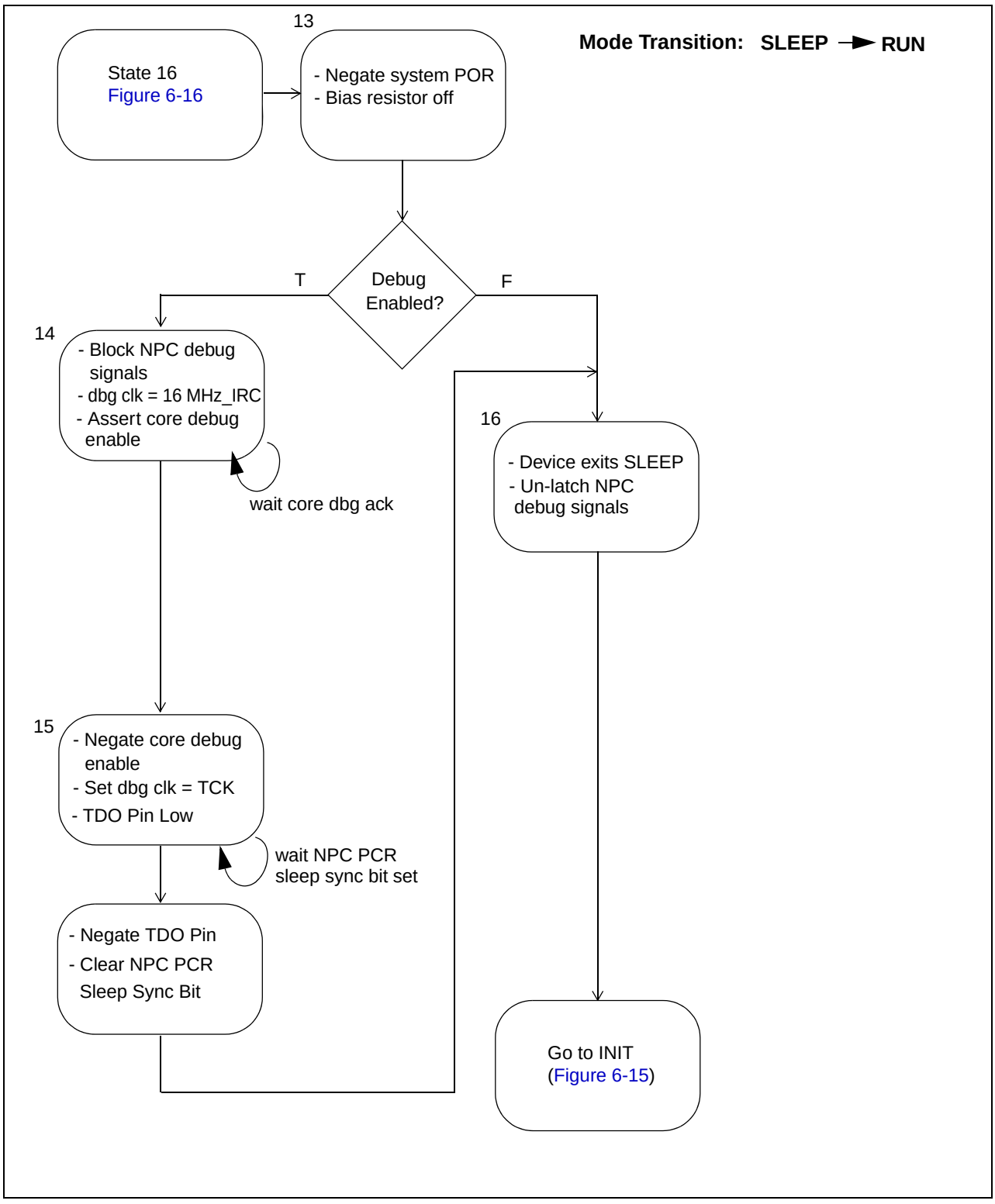


Figure 6-17. SLEEP Mode Transition Diagram (Part 2)

6.3.4.1 Sleep Mode Reset Operation

The reset controller in the SIU controls the normal reset sequences from POR, LVI, and other resets when the device is in RUN mode. The CRP controls reset operation for the device in sleep mode.

The external RESET pin is enabled in all modes. Assertion of the RESET pin or a POR during sleep mode causes the device to restart in RUN mode.

Upon power up from sleep mode, POR and reset is asserted to all logic that was powered down. The SIU processes the sleep recovery POR in the same manner as a normal POR. The RSR[PORS] bit in the SIU is set after the reset controller sequence completes. The CRP_PSCR[SLEEPF] bit is set in this case to indicate that the POR came from a sleep recovery.

NOTE

When powering up from sleep mode, the BOOTCFG pin is not read and the BAM boot sequence is bypassed since the Z6 and Z0 cores branch to the appropriate reset vector set in the CRP_Z0VEC and CRP_Z6VEC registers, assuming the core was active (not held in reset) prior to sleep mode entry.

6.3.5 Low-Power Wakeup

A POR, LVI12, or assertion of the external $\overline{\text{RESET}}$ pin causes exit from sleep mode as a reset condition, and not a wakeup. A POR or external reset is captured in the SIU Reset Status Register. All CRP registers are reset for a POR, but some like the CRP_RTCC are maintained for an external reset. Note that there are no internal reset sources (except POR and LVI12) active in sleep.

There are four methods for waking up the device from sleep mode:

- RTC counter match
- RTC counter rollover
- API counter match
- External pin transition

All wakeup methods are independently enabled. The RTC, RTC rollover, and API wakeup logic is discussed in Section 6.4, Real-Time Counter (RTC).

Wakeup from sleep can be enabled from transitions on as many as 32 external pins. External pin wakeup source selection is done in the CRP_PWKENH/L registers. To be used as a low-power mode wakeup, pins must be configured with the output buffer disabled in the SIU_PCR registers prior to entry into the low-power mode. During sleep mode, all pins (except the 32 wakeup pins and RESET) are put into a safe state with the input buffer, output buffer, and pull devices disabled. The wakeup pins input buffer enables will retain the state configured prior to entry into sleep mode with the output buffers and pull devices disabled.

External pin wakeup generation can be selected for either a rising edge event on the pin, falling edge, or both. The edge capture logic can be selectively clocked from either the 16 MHz IRC clock for faster wakeup, or the 128 kHz IRC clock for lower average power. For wakeup, the value of the Pin Assignment (PA) bitfield in the SIU_PCR does not matter. This enables a pin, such as a CAN receive pin, to wake up the device on a transition.

The corresponding CRP_PSCR[PWKSRCF] flag bit is set when a selected and enabled event occurs on an external pin wakeup source. An interrupt request can be generated for an external pin wakeup by setting the corresponding CRP_PSCR[PWKSRIE] bit. This interrupt request is pending once the device recovers from the previous low-power mode.

On exiting sleep mode, the PC value is loaded with the value contained in the CRP_Z6VEC or CRP_Z0VEC registers. The RECPTR register is a general purpose register which retains a value during sleep mode and thus may be used by software to hold a generic value used by recovery routines.

A block diagram for the external pin wakeup logic is given in Figure 6-18.

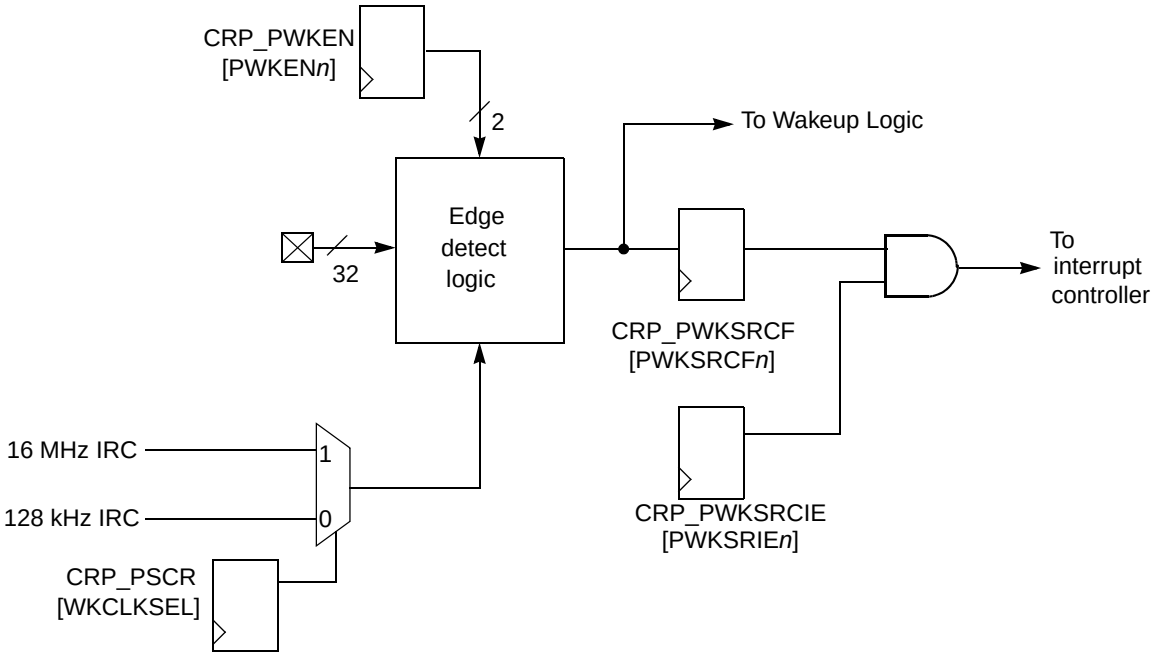


Figure 6-18. External Pin Wakeup Logic

6.3.5.1 Low Power Mode Debug Support

The CRP supports debug after exit from sleep mode for both Nexus and JTAG debug tools. This function is enabled by setting the NPC PCR[LP_DBG_EN] bit prior to entry into sleep modes.

On entry into sleep mode, if the NPC PCR[LP_DBG_EN] bit is set, the CRP sets the NPC PCR[SLEEP_SYNC] bit to inform the debug tool that sleep mode is being entered. The CRP waits for this bit to be cleared before proceeding into sleep mode. During sleep mode, most of the SOC is powered down, and the contents of the debug registers are lost. The CRP supports restoration of the debug registers on wakeup from sleep mode. The CRP latches the NPC PCR[LP_DBG_EN] bit when sleep mode is entered. On wakeup from sleep mode, if the latched bit is set, the CRP places both the Z0 and Z6 cores into debug mode. The CRP selects the 16 MHz IRC to clock the core debug logic, so the development tool does not need to drive a clock on the TCK pin at this point. Once both cores have acknowledged that they have entered debug mode, the CRP allows the TCK pin to drive the debug logic, enables the JTAG pins, and drives the assertion of the TDO pin.

NOTE

TDO is pulled high on entry into low power mode. It is driven low when the MCU wakes up.

The assertion of the TDO pin indicates to the debug tool that it can now restore the debug register contents via the JTAG interface. The Nexus pins cannot be used until the NPC configuration is restored. The TDO pin remains asserted until the debug tool sets the NPC PCR[SLEEP_SYNC] bit. At that point, TDO is negated, control of the pin given back to the JTAG controller, and the wakeup interrupt is asserted to the Z0 and Z6 cores. A block diagram of the SOC blocks and the connections between them to support debug on sleep wakeup is given in Figure 6-19.

NOTE

The CRP enables only the debug pins that were enabled prior to sleep mode entry.

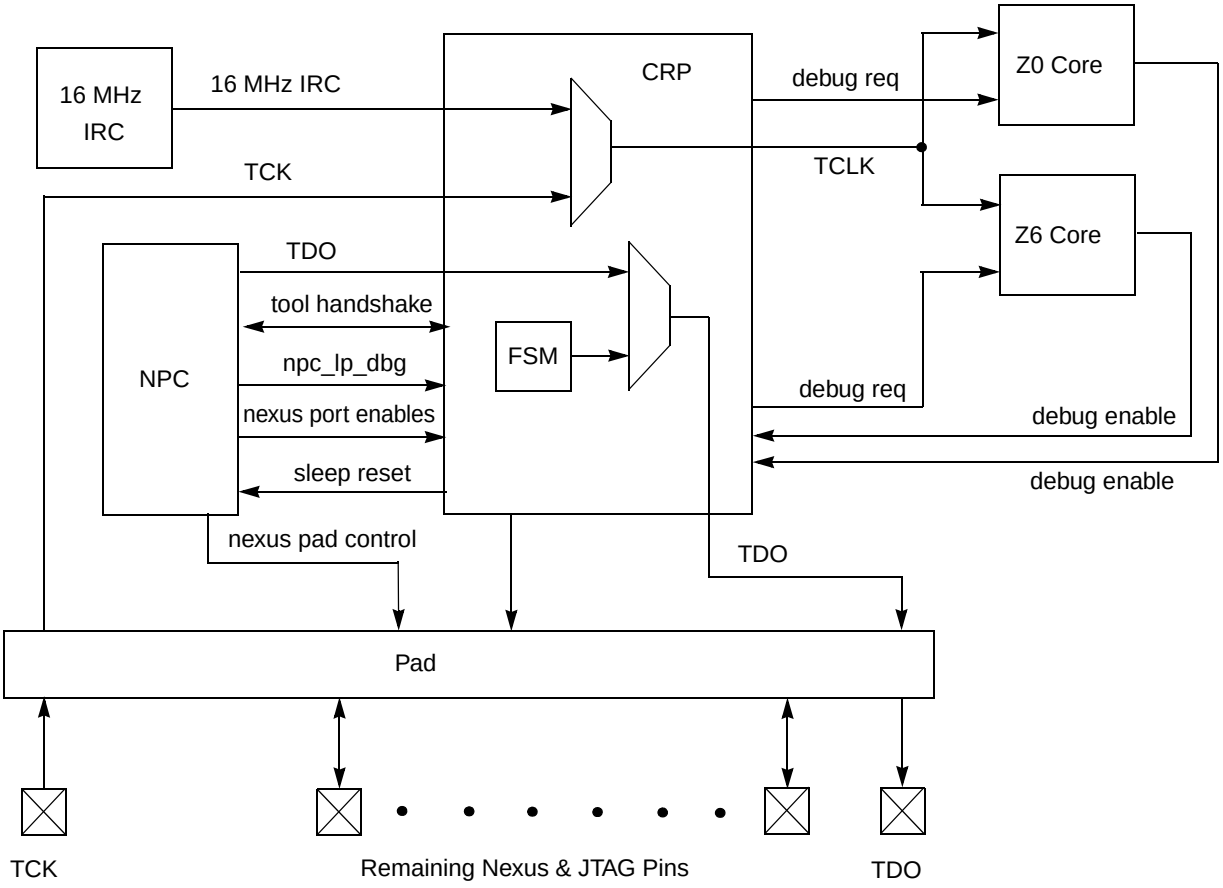


Figure 6-19. Sleep Mode Debug Block Integration

6.4 Real-Time Counter (RTC)

The RTC is a free-running counter used for time-keeping applications. The RTC may be configured to generate an interrupt at a pre-defined interval independent of the mode of operation. If in sleep mode when the RTC interval is reached, the RTC first generates a wakeup, and then asserts the interrupt request.

The RTC also supports an autonomous periodic interrupt function used to generate a periodic wakeup request to exit sleep mode or an interrupt request.

6.4.1 RTC Features

Features of the RTC include:

- 32-bit counter
- Four selectable counter clock sources:
  - 128 kHz IRC
  - 32 kHz_XTAL
  - 16 MHz IRC
  - 4 – 40 MHz XTAL (restricted to ≤ 8 MHz)
- Optional divide-by-512 prescaler and optional divide-by-32 prescaler connected in series in the clock path feeding the 32-bit counter
- 32-bit counter supports times up to greater than 1.5 months with 1 ms resolution
- 12-bit compare value to support interrupt intervals of 1 s up to greater than 1 hr with 1 s resolution
- RTC interrupt with interrupt enable
- Counter runs in all modes of operation
- RTC status and control register are reset only by POR RTCVAL and APIVAL can be updated anytime without disabling the clock
- RTC counter is reset when counter is disabled by software and by POR
- Autonomous periodic interrupt support includes:
  - 10-bit compare value to support wake-up intervals of 1.0 ms to 1 s
  - Wake-up logic has separate enable to support changing compare value while RTC running API interrupt with interrupt enable
  - Operates in all modes of operation
  - API compare value can be modified while RTC is running
- Optional interrupt for RTC match, API match, and RTC rollover
- RTC continues to count through all resets except POR, VDD12 LVI, VDD33 LVI, VDDSYN LVI, VDD5 Low LVI, and VDD5 LVI.

6.4.2 RTC Functional Description

The RTC consists of a 32-bit free-running counter enabled with CNTEN. (CNTEN when negated asynchronously resets the counter and synchronously enables the counter when enabled.) The value of the counter may be read via the RTCCNT register. Due to the clock synchronization, the RTCCNT value may actually represent a previous counter value.

The clock source to the counter is selected with CLKSEL and may be the 128 kHz IRC, the 32 kHz XTAL, 16 MHz IRC, or the 4 – 40 MHz XTAL (if restricted to ≤ 8 MHz). The clock path feeding the 32-bit counter can optionally be divided by the divide-by-512 prescaler or the divide-by-32 prescaler. Note that

the 32 kHz OSC must be enabled before being selected. The 32 kHz OSC is selected to give a more accurate wakeup than the 128 kHz IRC. (CNTEN must be disabled when the clock sources are switched.)

When the counter value for counter bits 10–21 match the 12-bit value in RTCVAL, then the RTCF interrupt flag is set (after proper clock synchronization). If the RTCIE interrupt enable bit is set, the RTC interrupt request is generated. The RTCF flag can be cleared by writing a 1 to RTCF. The RTCF supports interrupt requests in the range of 1 second to 4096 seconds (> 1 hr) with a 1 second resolution.

NOTE

RTCVAL and APIVAL can be updated at any time.

If there is a match while in sleep mode, and the CRP_PSCR[RTCWKEN] bit is set, then the RTC first generates a wakeup request to force a wakeup to run mode, then sets the RTCF flag. The RTC wakeup signal is captured in the CRP_PSCR[WKRTCF] flag bit.

A rollover wakeup and/or interrupt can be generated when the RTC transitions from a count of 0xFFFF_FFFF to 0x0000_0000. The rollover flag is enabled by setting the CRP_RTCC[ROVREN] bit. An RTC counter rollover with this bit and the CRP_PSCR[RTCOVRWKEN] bit set causes a wakeup from sleep mode. The rollover wakeup flag is captured in the CRP_PSCR[WKRLLOVRF] bit. An interrupt request is generated for an RTC counter rollover when both the CRP_RTCC[ROVREN] and CRP_RTCC[RTCIE] bits are set.

Setting APIEN enables the autonomous interrupt function. The 10 bit APIVAL selects the time interval for triggering an interrupt and/or wakeup event. Since the RTC is a free-running counter, the APIVAL is added to the current count to calculate an offset. When the counter reaches the offset count, a interrupt and/or wakeup request is generated. Then the offset value is recalculated and again retriggers a new request when the new value is reached. APIVAL (and RTCVAL) can be updated at any time. When a compare is reached, the APIF interrupt flag is set (after proper clock synchronization). If the APIIE interrupt enable bit is set, then the API interrupt request is generated. The APIF flag can be cleared by writing a 1 to APIF. If there is a match while in sleep mode, and the CRP_PSCR[APIWKEN] bit is set, then the API first generates a wakeup request to force a wakeup to run mode, then sets the APIF flag. The API wakeup flag is captured in the CRP_PSCR[WKAPIF] bit.

If the CRP_RTCC[FRZEN] bit is set, the RTC counter is frozen during debug mode.

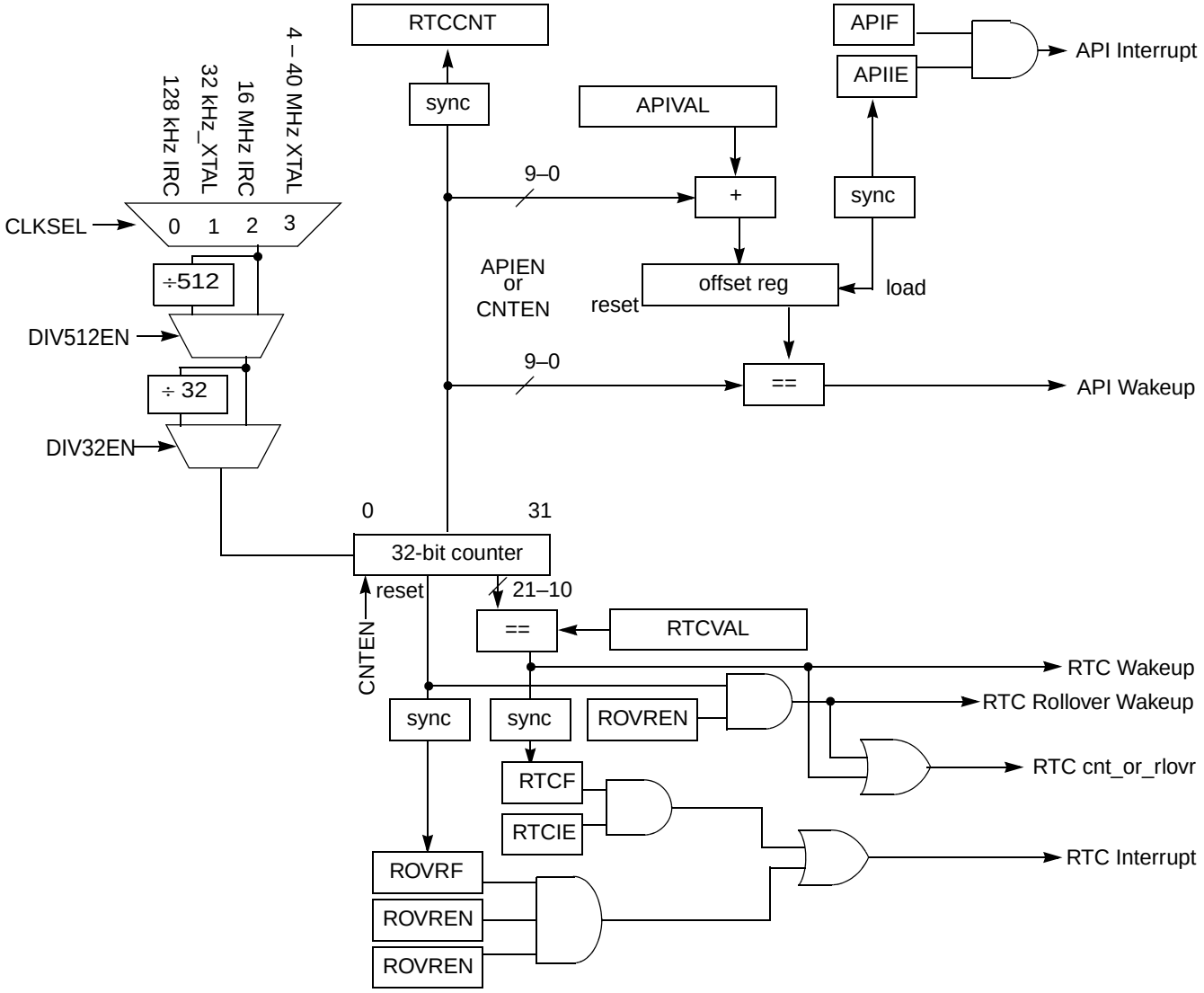


Figure 6-20. RTC/API Block Diagram

6.4.3 Register Description

The RTC registers control and monitor operation of the RTC. The registers that are relevant to the use of the RTC are as follows.

- RTC status and control register (Section 6.2.2.2, RTC Control Register (CRP_RTCC))
- RTC counter register (Section 6.2.2.4, RTC Counter Register (CRP_RTCCNT))

6.5 Power Supply Monitors

6.5.1 Power-On Reset (POR)

The internal Power On Reset (POR) monitors the main supply input voltage (V_{DDA}) and shall not release the internal reset line until V_{DDA} is above the de-assertion threshold. The POR is always enabled.

6.5.2 Low-Voltage Monitors (LVI)

The internal LVI circuits monitor when the voltage on the corresponding supply is lower than defined values, and either assert a reset or an interrupt. All LVI circuits are enabled in run mode. In sleep mode, LVI12 remains on. The LVIs also support hysteresis in the falling and rising trip points.

- LVI12—1.2 V supply
  - The LVI12 supply monitors V_{DD} and triggers a reset when it drops below the assert threshold of the LVI12.
- LVI33—3.3 V supply
  - The LVI33 monitors V_{DD33} and triggers a reset when it falls below the assert level.
- LVI33SYN—3.3 V V_{DDSYN} supply
  - The LVI33SYN monitors V_{DDSYN} and triggers a reset when it falls below the assert level.
- LVI5_VDDA—3.3 V – 5 V supply
  - The LVI_VDDA monitors V_{DDA} and triggers an interrupt or internal reset when it drops down below the assert level. LVI5_VDDA is automatically disabled when VRCSEL is low.
- LVI5L_VDDA— 3.3 V – 5 V supply
  - The LVIL_VDDA monitors V_{DDA} and triggers an internal reset when it drops down below the assert level.
- LVI5H_VDDA— 3.3 V – 5 V supply
  - The LVIH_VDDA monitors V_{DDA} and may be used to generate an internal interrupt when it drops down below the assert level. LVI5H_VDDA is automatically disabled when VRCSEL is low.

When a LVI5 trigger event occurs, the CRP_SOCSC[LVI5F] flag bit is set, and either a reset or an interrupt generated, depending on the configuration of the CRP_SOCSC[LVI5IE] and CRP_SOCSC[LVIRE] bits in the CRP. The CRP_SOCSC[LVI5RE] is always writable as long as the CRP_SOCSC[LVI5LOCK] bit is cleared. When CRP_SOCSC[LVI5LOCK] is set, then writes to CRP_SOCSC[LVI5RE] have no effect. The CRP_SOCSC[LVI5LOCK] bit is write-once and cleared only with POR.

There is no internal LVI monitoring of the individual V_{DDE} I/O segments.

Chapter 7
Frequency Modulated Phase-Locked Loop (FMPLL)

7.1 Introduction

The frequency modulated phase-locked loop (FMPLL) module is a frequency modulated phase-locked loop that has been optimized to generate voltage controlled oscillator (VCO) frequencies from 192 MHz – 600 MHz based on an input clock range of 4 MHz to 40 MHz. The frequency multiplication, output dividers and the frequency modulation waveform are register-programmable through a peripheral bus interface.

7.1.1 Block Diagram

A simplified block diagram of the FMPLL illustrates the functionality and interdependence of major blocks (see Figure 7-1). Shaded blocks represent analog circuit components that make up the core analog portion of the FMPLL. The complete FMPLL closed-loop system contains the feedback divider (EMFD) and output divider (ERFD), which are implemented with standard cell core logic elements. Refer to Section 7.4.3.3, PLL Normal Mode Without FM, for details on each sub-block.

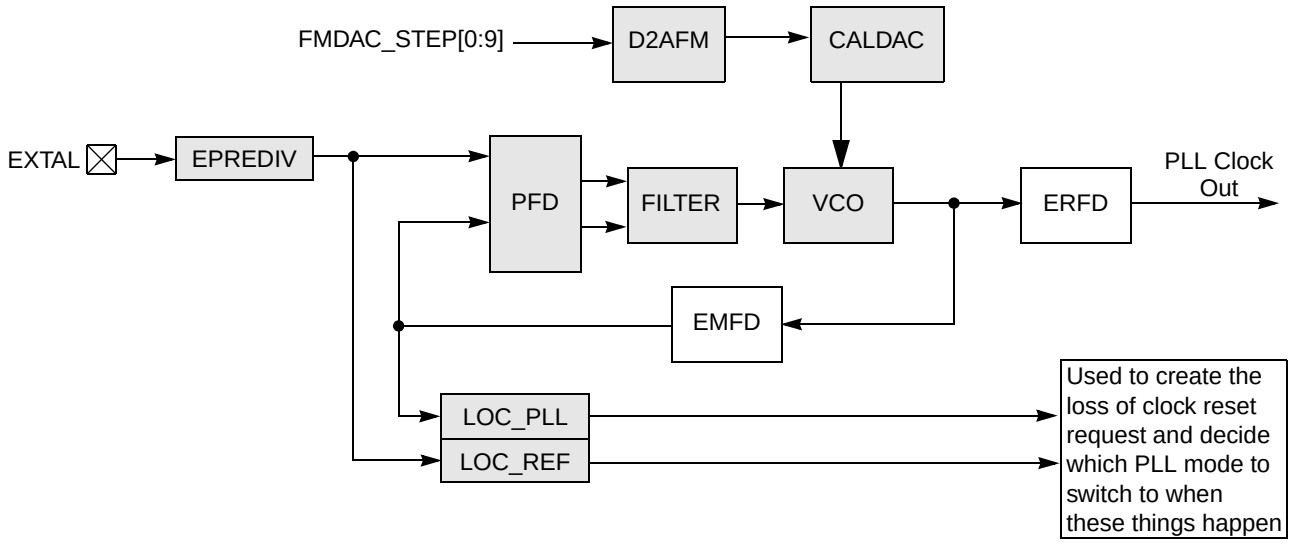


Figure 7-1. FMPLL Block Diagram

7.1.2 Features

The FMPLL has these major features:

- Input clock frequency range: 4 MHz to 40 MHz (EXTAL)

- Because the PXN20 uses a 16 MHz IRC as its default system clock, the FMPLL is put in PLL Off mode during reset, so that power dissipation is minimized by disabling the FMPLL until needed by the system.
- Programmable frequency multiplication factor settings generating VCO frequencies of 192 MHz – 600 MHz
- PLL Off mode (low-power mode)
- Register programmable output clock divider (ERFD)
- Programmable frequency modulation
  - Modulation applied as a triangle waveform
  - Peak-to-peak register programmable modulation depths of 0.5%, 1%, 1.5%, and 2% of the system frequency
  - Register programmable modulation rates of $F_{extal}/80$, $F_{extal}/40$, and $F_{extal}/20$
- Lock detect circuitry provides a signal indicating the FMPLL has acquired lock and continuously monitors the FMPLL output for any loss of lock
- Loss-of-clock circuitry monitors input reference and FMPLL output clocks with programmable ability to select a backup clock source as well as generate a reset or interrupt in the event of a failure

7.1.3 Modes of Operation

There are two main modes of FMPLL: PLL Off mode and normal mode. These modes are briefly described in this section.

When PLL Off mode is selected, the FMPLL is turned off and the end-system user must select a different source than the PLL output in SIU_SYSCLK[SYSCLKSEL]. The lock detector is not functional and does not indicate that the FMPLL is in a locked state. Frequency modulation is not available and the FMPLL is put into a low-power, idle state. This operating mode is described in Section 7.4.2, PLL Off Mode.

When normal mode is selected, the FMPLL is fully programmable. The FMPLL reference clock source can be a crystal oscillator or an external clock generator. The lock detector indicates the lock status of the FMPLL, and frequency modulation of the output clock can be enabled. This operating mode is described in Section 7.4.3, Normal Mode.

7.2 External Signal Description

Refer to Table 3-1 and Section 3.4, Detailed Signal Description, for detailed signal descriptions.

7.3 Memory Map and Registers

This section provides a detailed description of the FMPLL registers.

7.3.1 Module Memory Map

Table 7-1 shows the FMPLL memory map. The address of each register is given as an offset to the FMPLL base address. Registers are listed in address order, identified by complete name and mnemonic, and lists the type of accesses allowed.

Table 7-1. FMPLL Memory Map

Offset from FMPLL_BASE_ADDR (0xFFFF_0000)	Register	Access	Reset Value	Section/Page
0x0000	Reserved			
0x0004	SYNSR—FMPLL synthesizer status register	R/W	0x0000_0000	7.3.2.1/7-3
0x0008	ESYNCR1—FMPLL enhanced synthesizer control register 1	R/W	0x8000_0030	7.3.2.2/7-5
0x000C	ESYNCR2—FMPLL enhanced synthesizer control register 2	R/W	0x0000_0003	7.3.2.3/7-7
0x0010–0x0014	Reserved			

7.3.2 Register Descriptions

This section lists the FMPLL registers in address order and describes the registers and their bit fields.

7.3.2.1 FMPLL Synthesizer Status Register (SYNSR)

Offset: FMPLL_BASE_ADDR + 0x0004 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	LOLF	LOC	MODE	PLL SEL	PLL REF	LOCKS	LOCK	LOCF	CAL DONE	CAL PASS
W							w1c							w1c		
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 7-2. FMPLL Synthesizer Status Register (SYNSR)

Table 7-2. SYNSR Register Field Descriptions

Field	Description
LOLF	Loss- of-Lock Flag. This bit provides the interrupt request flag. To clear the flag, write a 1 to the bit. Writing 0 has no effect. This flag will not be set, and an interrupt will not be requested, if the loss-of-lock condition was caused by a system reset, enabling of frequency modulation, or a write to the ESYNCR1 which modifies the ESYNCR1[EMFD] bits. If the flag is set due to a system failure, writing the ESYNCR1[EMFD] bits or enabling FM does not clear the flag. Assert reset to clear the flag. If lock is reacquired, the bit remains set until either a write 1 or reset is asserted. 0 Interrupt service not requested. 1 Interrupt service requested.

Table 7-2. SYNSR Register Field Descriptions (continued)

Field	Description
LOC	Loss-Of-Clock Status. The LOC bit is an indication of whether a loss-of-clock condition is present when operating in normal PLL mode. If LOC = 0, the system clocks are operating normally. If LOC = 1, the system clocks have failed due to a reference failure or a PLL failure. If the read of the LOC bit and the loss-of-clock condition occur simultaneously, the bit does not reflect the current loss-of-clock condition. If a loss-of-clock condition occurs that sets this bit and the clocks later return to normal, this bit is cleared. LOC is always zero in PLL Off mode. 0 Clocks are operating normally. 1 Clocks are not operating normally.
MODE	Clock Mode. The state of this bit, along with PLLSEL and PLLREF, indicates which clock mode the PLL is operating in (see Table 7-3). The value of ESYNCR1[CLKCFG0] is reflected in this location. 0 PLL Off mode. 1 PLL clock mode.
PLLSEL	PLL Mode Select. The state of this bit, along with MODE and PLLREF, indicates which mode the PLL operates in (see Table 7-3). This bit is cleared in PLL Off mode. The value of ESYNCR1[CLKCFG1] is reflected in this location. 0 PLL Off mode. 1 Normal PLL mode.
PLLREF	PLL Clock Reference Source. The state of this bit, along with MODE and PLLSEL, indicates which reference source has been chosen for normal PLL mode (see Table 7-3). This bit is cleared in PLL Off mode. The value of ESYNCR1[CLKCFG2] is reflected in this location. 0 External clock reference chosen. 1 Crystal clock reference chosen. **Note:** The user must also use the XOSC bit in the CRP_CLKSRC register to enable the 4 – 40 MHz oscillator.
LOCKS	Sticky PLL Lock Status Bit. The LOCKS bit is a sticky indication of PLL lock status. LOCKS is set by the lock detect circuitry when the PLL acquires lock after: 1) a system reset, or 2) a write to the ESYNCR2 which modifies the ESYNCR2[EMFD] bits, or 3) frequency modulation is enabled. Whenever the PLL loses lock, LOCKS is cleared. LOCKS remains cleared after the PLL re-locks, until one of the three conditions occurs. Furthermore, if the LOCKS bit is read when the PLL simultaneously loses lock, the bit does not reflect the current loss-of-lock condition. If operating in PLL Off mode, LOCKS remains cleared after reset. 0 PLL has lost lock since last system reset, a write to ESYNCR1 to modify the ESYNCR1[EMFD] bit field, or frequency modulation enabled 1 PLL has not lost lock since last system reset, a write to ESYNCR1 to modify the ESYNCR1[EMFD] bit field, or frequency modulation enabled
LOCK	PLL Lock Status Bit. The LOCK bit indicates whether the PLL has acquired lock. PLL lock occurs when the synthesized frequency matches to within approximately 0.75% of the programmed frequency. The PLL loses lock when a frequency deviation of greater than approximately 1.5% occurs. If the LOCK bit is read when the PLL simultaneously loses lock or acquires lock, the bit does not reflect the current condition of the PLL. If operating in PLL Off mode, LOCK remains cleared after reset. 0 PLL is unlocked 1 PLL is locked
LOCF	Loss-of-Clock Flag. This bit provides the interrupt request flag. To clear the flag, write a 1 to the bit. Writing 0 has no effect. Asserting reset clears the flag. If clocks return to normal after the flag has been set, the bit remains set until cleared by either writing 1 or asserting reset. A loss-of-clock condition can only be detected if LOCEN = 1. 0 Interrupt service not requested. 1 Interrupt service requested.

Table 7-2. SYNSR Register Field Descriptions (continued)

Field	Description
CALDONE	Calibration Complete. The CALDONE bit is an indication of whether the calibration sequence has been completed since the last time modulation was enabled. If CALDONE = 0 then the calibration sequence is in progress or modulation is disabled. If CALDONE = 1 then the calibration sequence has been completed, and frequency modulation is operating. 0 Calibration not complete. 1 Calibration complete.
CALPASS	Calibration Passed. The CALPASS bit tells whether the calibration routine was successful. If CALPASS = 1 and CALDONE = 1 then the routine was successful. If CALPASS = 0 and CALDONE = 1, then the routine was unsuccessful. When the calibration routine is initiated the CALPASS is asserted. CALPASS remains asserted until modulation is disabled by clearing the EDEPTH bits in the ESYNCR2 register or a failure occurs within the FMPLL calibration sequence. 0 Calibration unsuccessful, 1 Calibration successful. If calibration is unsuccessful, then actual depth is not guaranteed to match the desired depth

Table 7-3. System Clock Status Per Mode

MODE	PLLSEL	PLLREF	Clock Mode
0	X	X	PLL Off mode
1	0	0	Reserved
1	1	0	Normal PLL mode with external clock reference
1	1	1	Normal PLL mode with crystal clock reference

NOTE

If LOLF has been set previously (due to an unexpected loss of lock condition) and then cleared (by writing a 1), a change of the MFD, PREDIV or DEPTH fields can cause the LOLF to be set again which can trigger an interrupt request if LOLIRQ bit is set. In addition, changing the RATE bit will also set the LOLF regardless of previous conditions.

The Loss of Lock Interrupt Request enable in the Synthesizer Control Register (FMPLL_SYNCR[LOLIRQ]) should be cleared before any change to the multiplication factor (MFD), PREDIV, modulation depth (DEPTH), or modulation rate (RATE) to avoid unintentional interrupt requests. After the PLL has locked (LOCK=1), LOLF should be cleared (by writing a 1) and LOLIRQ may be set again if required.

7.3.2.2 FMPLL Enhanced Synthesizer Control Register 1 (ESYNCR1)

This is one of two FMPLL synthesizer control registers that are used to access enhanced features in the FMPLL. The bit fields in the ESYNCR1 behave as described in Figure 7-3.

Offset: FMPLL_BASE_ADDR + 0x0008 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	1	CLKCFG[0:2]			0	0	0	0	0	0	0	0	EPREDIV			
W																
Reset	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	EMFD							
W																
Reset	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	

Figure 7-3. FMPLL Enhanced Synthesizer Control Register 1 (ESYNCR1)

Table 7-4. ESYNCR1 Register Field Descriptions

Field	Description
bit 0	Reserved. Note: This bit is set to 1 on reset and always reads as 1.
CLKCFG	Clock Configuration. The CLKCFG[0:2] bits are writable versions of the MODE, PLLSEL, and PLLREF bits in the SYNSR. These change the clock mode, after reset has negated, via software. CLKCFG[0:2] map directly to MODE, PLLSEL, and PLLREF to control the system clock mode (see Table 7-3). **Note:** CLKCFG[0:2] = 0b101 can produce an unpredictable clock output. **Note:** The ESYNCR2[LOLRE] and ESYNCR2[LOCRE] should be set to 0 before changing the PLL mode, so that a reset is not immediately generated when CLKCFG[0:2] is written.
EPREDIV	Enhanced Pre-Divider. The EPREDIV bits control the value of the divider on the input clock. The output of the pre-divider circuit generates the reference clock to the PLL analog loop. The decimal equivalent of the EPREDIV binary number is substituted into the equation from Table 7-10. **Note:** Setting EPREDIV to any of the invalid states in Table 7-4 causes the PLL to produce an unpredictable output clock. The output frequency of the divider must equal f_{pllref} (see the *PXN20 Microcontroller Data Sheet*). When the EPREDIV bits are changed, the PLL immediately loses lock. If the EPREDIV bits are changed during FM calibration, the current calibration sequence is terminated and the DEPTH bits are cleared. The PLL re-locks to the new EPREDIV value. Modulation must be re-enabled manually. To prevent an immediate reset, clear the LOLRE bit before writing the EPREDIV bits. In PLL bypass mode, the EPREDIV bits have no effect.
EMFD	Enhanced Multiplication Factor Divider. The EMFD bits control the value of the divider in the PLL feedback loop. The value specified by the EMFD bits establish the multiplication factor applied to the reference frequency. The decimal equivalent of the EMFD binary number is substituted into the equation from Table 7-11 for F_{sys} to determine the equivalent multiplication factor. The range of settings is $32 \leq EMFD \leq 132$. **Note:** EMFD values less than 32 and greater than 132 are invalid and cause the PLL to produce an unpredictable clock output. The VCO frequency must be within the f_{VCO} specification (see the *PXN20 Microcontroller Data Sheet*). When the EMFD bits are changed, the PLL loses lock.If the EMFD bits are changed during FM calibration, the current calibration sequence is terminated and the DEPTH bits are cleared. The PLL re-locks to the new EMFD value and you must manually re-enable modulation. To prevent an immediate reset, clear the LOLRE bit before writing the EMFD bits. In PLL Off mode, the EMFD bits have no effect. Table 7-6 shows the available divide ratios.

Table 7-5. Pre-divider Ratios

EPREDIV	Input Divide Ratio (EPREDIV+1)
0000	1 (default for PXN20)
0001	2
0010	3
0011	4
0100	5
0101	6
0110	Invalid
0111	8
1000	Invalid
1001	10
1010–1111	Invalid

Table 7-6. Feedback Divide Ratios

EMFD	Feedback Divide Ratio (EMFD+16)
0000_0000–0001_1111	Invalid
0010_0000	48
0010_0001	49
0010_0010	50
0010_0011	51
0010_0100	52
0010_0101	53
. . 0011_0000 . .	. . 64 (default for PXN20) . .
1000_0100	148
1000_0101–1111_1111	Invalid

7.3.2.3 FMPLL Enhanced Synthesizer Control Register 2 (ESYNCR2)

This is the second of two enhanced versions of the FMPLL synthesizer control register used to access enhanced features in the FMPLL. The bit fields in the ESYNCR2 behave as described in Figure 7-4.

Offset: FMPLL_BASE_ADDR + 0x000C Access: User read/write

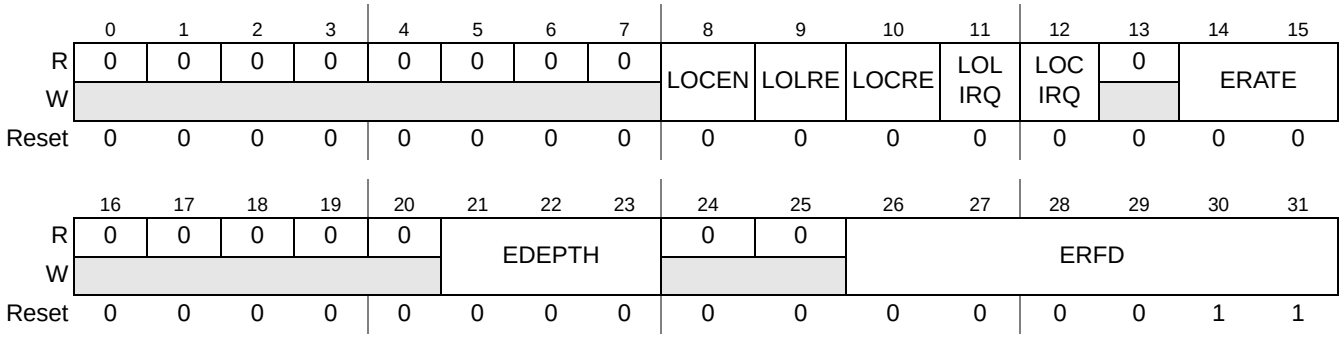


Figure 7-4. FMPLL Enhanced Synthesizer Control Register 2 (ESYNCR2)

Table 7-7. ESYNCR2 Field Descriptions

Field	Description
LOCEN	Loss-of-Clock Enable. The LOCEN bit determines whether the loss-of-clock function is operational along with backup clock modes, and interrupt and reset functions. See Section 7.4.3.2, Loss-of-Clock Detection, for more information. In PLL Off mode, this bit has no effect. LOCEN does not affect the loss-of-lock circuitry. 0 Loss-of-clock disabled. 1 Loss-of-clock enabled.
LOLRE	Loss-of-Lock Reset Enable. The LOLRE bit determines how the integration module handles a loss-of-lock indication. See Section 7.4.3.1, PLL Lock Detection, for more information. When operating in normal PLL mode, the PLL must be locked before setting the LOLRE bit. Otherwise reset is immediately asserted. The LOLRE bit has no effect in PLL Off mode. 0 Assert reset on loss of lock is disabled. 1 Assert reset on loss of lock.
LOCRE	Loss-of-Clock Reset Enable. The LOCRE bit determines how the integration module handles a loss-of-clock condition when LOCEN is equal to 1. LOCRE has no effect when LOCEN is equal to 0. If the LOCF bit in the SYNSR indicates a loss-of-clock condition, setting the LOCRE bit causes an immediate reset. The LOCRE bit has no effect in PLL Off mode. 0 Assert reset on loss of clock is disabled. 1 Assert reset on loss of clock.
LOLIRQ	Loss-of-Lock Interrupt Request. The LOLIRQ bit determines how the integration module handles a loss-of-lock indication. See Section 7.6.1, Loss-of-Lock Interrupt Request, for more information. When operating in normal mode, the PLL must be locked before setting the LOLIRQ bit. Otherwise an interrupt is immediately requested. The LOLIRQ bit has no effect in PLL Off mode. 0 Request interrupt is disabled. 1 Request interrupt.
LOCIRQ	Loss- of-Clock Interrupt Request. The LOCIRQ bit determines how the integration module handles a loss-of-clock condition when LOCEN = 1. LOCIRQ has no effect when LOCEN = 0. If the LOCF bit in the SYNSR indicates a loss-of-clock condition, setting (or having previously set) the LOCIRQ bit causes an interrupt request. The LOCIRQ bit has no effect in PLL Off mode. 0 Request interrupt on loss of clock is disabled. 1 Request interrupt on loss of clock.

Table 7-7. ESYNCR2 Field Descriptions (continued)

Field	Description
ERATE	Enhanced Modulation Rate. The ERATE bits control the rate of frequency modulation applied to the system frequency. Table 7-8 shows the allowable modulation rates. **Note:** The PLL modulation rate must be within the f_{MOD} specification (see the *PXN20 Microcontroller Data Sheet*).
EDEPTH	Enhanced Modulation Depth. The EDEPTH bit field controls the frequency modulation depth and enables the frequency modulation. When programmed to a value other than 0x0, the frequency modulation is automatically enabled. Table 7-9 shows are the programmable frequency deviations from the system frequency. When the depth value is changed to a value other than 0x0, the calibration sequence is reinitialized.
ERFD	Enhanced Reduced Frequency Divider. The ERFD bits control a divider at the output of the PLL. The value specified by the ERFD bits establish the divisor applied to the PLL frequency. The ERFD divides the output clock by the quantity (ERFD + 1). Even-numbered ERFD settings, which would result in odd divide ratios, are not allowed. The decimal equivalent of the ERFD binary number is substituted into the equation from Table 7-11. **Note:** The ERFD divides the output clock by the quantity (ERFD + 1). Even numbered ERFD settings, which would result in odd divide ratios, are invalid and cause the PLL to produce an unpredictable output clock. The PLL output clock must be within the f_{PLL} specification (see the *PXN20 Microcontroller Data Sheet*). Changing the ERFD bits does not affect the PLL, hence, no re-lock delay is incurred. Resulting changes in clock frequency are synchronized to the next falling edge of the current system clock. These bits should be written only when the lock bit (LOCK) is set, to avoid surpassing the allowable system operating frequency. In PLL Off mode, the ERFD bits have no effect. The available output divider ratios are given in Table 7-10.

Table 7-8. Programmable Modulation Rates

ERATE	Modulation Rate (Hz)
00	$F_{mod} = F_{extal}/80$
01	$F_{mod} = F_{extal}/40$
10	$F_{mod} = F_{extal}/20$
11	Invalid

Table 7-9. Programmable Modulation Depths

EDEPTH	Modulation Depth (% of F_{sys})
000	0
001	0.25% – 0.5%
010	0.75% – 1.0%
011	1.25% – 1.5%
100	1.75% – 2.0%
101 – 111	Reserved

Table 7-10. Output Divide Ratios

ERFD	Output Divide Ratio (ERFD+1)
00_0000	1
00_0001	2
00_0010	Invalid
00_0011	4 (default value for PXN20)
00_0100	Invalid
00_0101	6
00_0110	Invalid
00_0111	8
. . .	. . .
11_1100	Invalid
11_1101	62
11_1110	Invalid
11_1111	64

7.4 Functional Description

The FMPLL module contains the frequency modulated phase lock loop (FMPLL), enhanced frequency divider (ERFD), enhanced synthesizer control registers (ESYNCR1 and ESYNCR2), synthesizer status register (SYNSR), and clock/PLL control logic. The block also contains a reference frequency pre-divider controlled by the EPREDIV bits in the ESYNCR1. This enables the user to use a high frequency crystal or external clock generator and obtain finer frequency synthesis resolution than would be available if the raw input clock were used directly by the analog loop. For the remainder of this chapter, the term “reference frequency” and the symbol F_{ref} indicate the output of the pre-divider circuit. This is the clock on which frequency multiplication is performed.

7.4.1 General

At reset, the system clock is driven by the internal oscillator (16 MHz IRC) and the module is in PLL Off mode. After reset, software can change the PLL mode (see Section 7.5.1, Clock Mode Selection).

Table 7-11 shows the PLL-clock to input-clock frequency relationships for the available clock modes.

Table 7-11. Clock-Out vs. Clock-In Relationships

Clock Mode	Frequency Equation
Normal PLL Mode	$F_{sys} = \frac{F_{extal} \bullet (EMFD + 16)}{(EPREDIV + 1)(ERFD + 1)}$

7.4.2 PLL Off Mode

When PLL Off mode is selected, the PLL is turned off. Either the 16 MHz IRC must be selected as the system clock, or the user must supply an external clock or crystal on the EXTAL pin and select that clock source before entering PLL off mode. The selected clock is directly used to produce the various system clocks. Refer to *PXN20 Microcontroller Data Sheet* for external clock input requirements. In PLL Off mode, the analog portion of the PLL is disabled, the frequency modulation capability is not available, and no clocks are generated at the PLL output. The pre-divider is bypassed and has no effect on the system clock frequency in PLL Off mode.

7.4.3 Normal Mode

When normal PLL mode is selected, the PLL is fully programmable. The PLL can synthesize frequencies ranging from 48× to 148× the reference frequency of the output of the predivider. with or without frequency modulation enabled. The post-divider is capable of reducing the PLL clock frequency without forcing a re-lock. The PLL reference can be a crystal oscillator reference or an external clock reference. This clock is divided by the pre-divider circuit to create the PLL reference clock.

7.4.3.1 PLL Lock Detection

The lock detect logic monitors the reference frequency and the PLL feedback frequency to determine when frequency lock has been achieved. Phase lock is inferred by the frequency relationship, but is not guaranteed. The PLL lock status is reflected in the LOCK status bit in the SYNSR. A sticky lock status indication, LOCKS, is also provided.

The lock detect function uses two counters, which are clocked by the reference and PLL feedback respectively. When the reference counter has counted N cycles, the feedback counter’s count is compared. If the feedback counter has also counted N cycles, the process is repeated for N + K counts. Then if the two counters’ counts match, the lock criteria is relaxed by one count and the system is notified that the PLL has achieved frequency lock. After three successful compares, the tolerance is relaxed.

After lock has been detected, the lock circuitry continues to monitor the reference and feedback frequencies using the alternate count and compare process. If the counters do not match at any comparison time, then the LOCK status bit is cleared to indicate that the PLL has lost lock. At this point, the lock criteria is tightened and the lock detect process is repeated.

The alternate count sequences prevent false lock detects due to frequency aliasing while the PLL tries to lock. Alternating between a tight and relaxed lock criteria prevents the lock detect function from randomly toggling between locked and not locked status due to phase sensitivities. Figure 7-5 illustrates the sequence for detecting locked and not-locked conditions.

When the frequency modulation is enabled, the loss of lock continues to function as described but with the lock and loss of lock criteria reduced to ensure that false loss of lock conditions are not detected.

In PLL Off mode, the PLL cannot lock because the PLL is disabled.

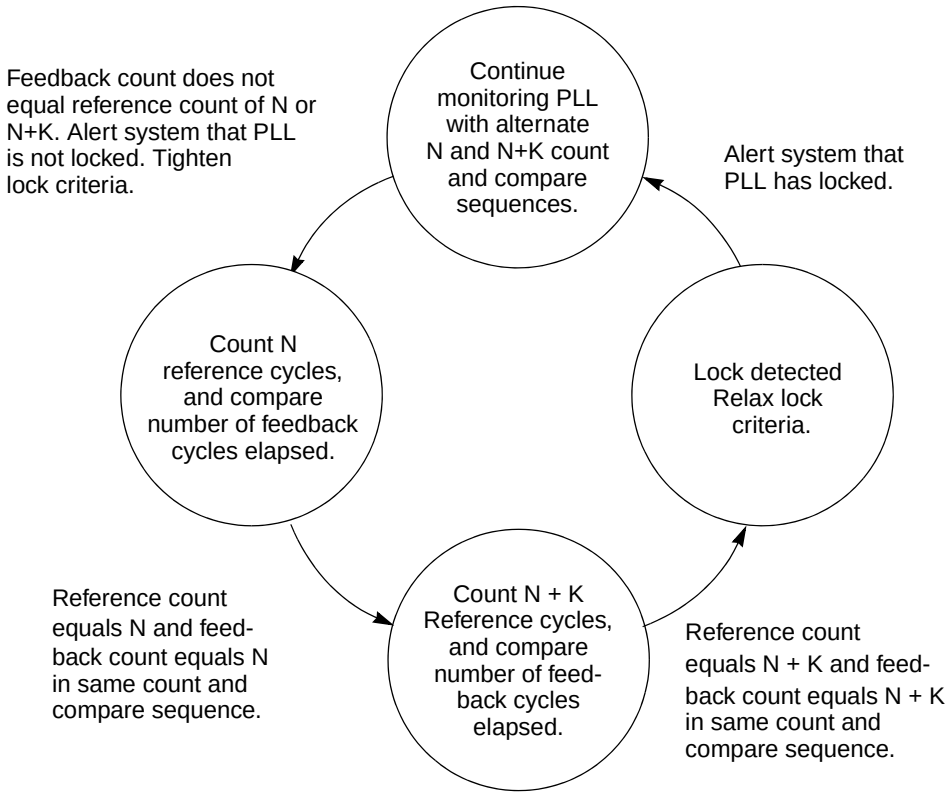


Figure 7-5. Lock Detect Sequence

After the PLL acquires lock after reset, the LOCK and LOCKS status bits are set. If the EPREDIV or EMFD are changed, or if an unexpected loss-of-lock condition occurs, the LOCK and LOCKS status bits are negated. While the PLL is in an unlocked condition, the system clocks continue to be sourced from the PLL as the PLL attempts to re-lock. Consequently, during the re-locking process, the system clock frequency is not well defined and may exceed the maximum system frequency violating the system clock timing specifications. Because of this condition, use of the loss-of-lock reset function is recommended.

After the PLL has re-locked, the LOCK bit is set. The LOCKS bit remains cleared if the loss of lock was unexpected. The LOCKS bit is set to one when the loss of lock was caused by changing the EPREDIV or EMFD fields.

7.4.3.2 Loss-of-Clock Detection

When enabled by the LOCEN bit in the ESYNCR2, the loss-of-clock (LOC) detection circuit monitors the input clocks to the phase/frequency detector (PFD) (see Figure 7-1). When the reference or feedback clock frequency falls below a minimum frequency, the LOC circuitry considers the clock to have failed and a loss-of-clock status is reflected by the sticky LOCF bit, and non-sticky LOC bit in the SYNSR. See *PXN20 Microcontroller Data Sheet* for the minimum clock frequency. In PLL Off mode, the loss-of-clock circuitry is disabled.

Depending which clock source has failed, the LOC circuitry switches the PLL output clock's source to the remaining operational clock if enabled by LOCEN. The PLL output clocks are derived from the alternate clock source until reset is asserted. The alternate clock source used is dependent on whether the LOC is

caused by a reference clock failure or a PLL failure. If the reference fails, the PLL goes out of lock and into self-clocked mode (SCM) (see Table 7-12). The PLL remains in SCM until the next reset. When the PLL is operating in SCM, the PLL runs open loop at a default VCO frequency. The RFD will set to divide-by-4 to ensure the clock presented to the system is well below the maximum allowable frequency for the device. If the loss-of-clock condition is due to a PLL failure (i.e., loss of feedback clock), the PLL reference becomes the system clock source until the next reset, even if the PLL regains itself and re-locks.

Table 7-12. Loss-of-Clock Summary

Clock Mode	System Clock Source before Failure	REFERENCE FAILURE Alternate Clock Selected by LOC Circuitry until Reset	PLL FAILURE Alternate Clock Selected by LOC Circuitry until Reset
PLL	PLL	PLL self-clocked mode	PLL reference
PLL Off	Ext. Clock(s)	None	NA

Note: The LOC circuit monitors the inputs to the PFD: reference and feedback clocks (see Figure 7-1).

A special loss-of-clock condition occurs when both the reference and the PLL fail. The failures may be simultaneous or the PLL may fail first. In either case, the reference clock failure takes priority and the PLL attempts to operate in SCM. If successful, the PLL remains in SCM until the next reset. During SCM, modulation is always disabled. If the PLL cannot operate in SCM, the system remains static until the next reset. If a loss-of-clock reset is enabled, then the reset switches the system clock over to the 16 MHz IRC (and shuts off the PLL).

7.4.3.3 PLL Normal Mode Without FM

In PLL mode, the system clocks are synthesized by the FMPLL by multiplying up the reference clock frequency. It is critical that the system clock frequency remain within the range for the device (see *PXN20 Microcontroller Data Sheet*). The output of the FMPLL can be divided down in powers ranging from 2 to 128 to reduce the system frequency with the ERFD. The ERFD is not contained in the feedback loop of the PLL, so changing the ERFD bits does not affect FMPLL operation. Finally, the PLL can be frequency modulated to reduce electromagnetic interference often associated with clock circuitry. Figure 7-1 shows the overall block diagram for the PLL. Each of the major blocks is discussed briefly in the following sections.

7.4.3.3.1 Phase/Frequency Detector

The phase/frequency detector (PFD) is a dual-latch phase-frequency detector. It compares both the phase and frequency of the reference clock and the feedback clock. The reference clock comes from the crystal oscillator or an external clock source. The feedback clock comes from the VCO output divided down by the EMFD in normal PLL mode.

When the frequency of the feedback clock equals the frequency of the reference clock (i.e., the PLL is frequency locked), the PFD pulses the UP or DOWN signals depending on the relative phase of the two clocks. If the falling edge of the reference clock leads the falling edge of the feedback clock, then the UP signal is pulsed. If the falling edge of the feedback clock leads the falling edge of the reference clock, then the DOWN signal is pulsed. The width of these pulses relative to the reference clock is dependent on how

much the two clocks lead or lag each other. After phase lock is achieved, the PFD continues to pulse the UP and DOWN signals for a very short duration during each reference clock cycle. These short pulses force the PLL to continually update and prevent a frequency drift phenomena referred to as "dead-banding." Dead-band describes the minimum amount of phase error between the reference and feedback clocks that a phase detector cannot correct.

7.4.3.3.2 Charge Pump/Loop Filter

Operation of the charge pump is controlled by the UP and DOWN signals from the PFD. They control whether the charge pumps apply or remove charge, respectively, from the loop filter.

7.4.3.3.3 VCO

The voltage into the VCO controls the frequency of its output. The frequency-to-voltage relationship (VCO gain) is positive.

7.4.3.3.4 EMFD

The MFD divides down the output of the VCO and feeds it back to the PFD. The PFD controls the VCO frequency (via the charge pump and loop filter) such that the reference and feedback clocks have the same frequency and phase. Thus, the input to the MFD, which is also the output of the VCO, is at a frequency that is the reference frequency multiplied by the same amount the MFD divides by. For example, if the MFD divides the VCO frequency by 48, then the PLL is frequency locked when the VCO frequency is 48 times the reference frequency. The presence of the MFD in the loop allows the PLL to perform frequency multiplication, or synthesis.

7.4.3.3.5 Programming System Clock Frequency

In normal PLL clock mode, the default system frequency is determined by the default EPREDIV, EMFD, and ERFD values.

When programming the PLL, do not violate the maximum system clock frequency or max/min VCO frequency specifications. Based on the desired system clock frequency, EPREDIV, EMFD, and ERFD must be calculated for the given crystal or external reference frequency. See *PXN20 Microcontroller Data Sheet* for the max/min VCO frequency range and the maximum allowable system frequency.

Frequency modulation should be disabled prior to changing the EPREDIV, EMFD, or RATE bit fields. After enabling frequency modulation a new calibration sequence is performed. A change to EPREDIV, EMFD, DEPTH, or RATE while modulation is enabled invalidates the previous calibration results.

Use these directions to accommodate the frequency overshoot that occurs when the EPREDIV or EMFD bits are changed. If frequency modulation is going to be enabled the maximum allowable frequency must be reduced by the programmed ΔF_m.

1. Determine the appropriate value for the EPREDIV, EMFD, and ERFD fields in the synthesizer control register(s), remember to include the ΔF_m if frequency modulation is to be enabled. The amount of jitter in the system clocks can be minimized by selecting the maximum EMFD factor that can be paired with an ERFD factor to provide the desired frequency. The maximum EMFD value that can be used is determined by the VCO and EMFD range.

2. Write a value of ERFD = ERFD (from step 1) + 1 to the ERFD field of the ESYNCR2. Not increasing the ERFD when changing the EPREDIV or EMFD could subject the device to clock frequencies beyond the range specified for the device due to the PLL's unlocked state.
3. If frequency modulation is currently enabled, disable it by writing 00 to the EDEPTH field of the ESYNCR2.
4. If programming the EPREDIV and/or EMFD, write the value(s) determined in step 1 to the appropriate field(s) in the ESYNCR1.
5. Monitor the synthesizer lock bit (LOCK) in the synthesizer status register (SYNSR). When the PLL achieves lock, write the ERFD value determined in step 1 to the ERFD field of the ESYNCR2. This changes the system clocks frequency to the desired frequency. If frequency modulation is desired, leave ERFD programmed to ERFD + 1 until after completing the steps in Section 7.4.3.4.2, Programming System Clock Frequency With Frequency Modulation.
6. If frequency modulation was enabled initially, it can be re-enabled following the steps listed in Section 7.4.3.4.2, Programming System Clock Frequency With Frequency Modulation.

7.4.3.4 PLL Normal Mode With Frequency Modulation

In normal PLL clock mode, frequency modulation is not enabled in the default synthesis mode. When frequency modulation is enabled two parameters must be set to generate the desired level of modulation. The parameters to be programmed are the RATE and DEPTH bit fields of the ESYNCR2 register. The RATE bit controls the frequency of modulation, F_{mod}. The DEPTH bits work to control the modulation depth, F_m. The available modulation rates and depths are given in Table 7-8 and Table 7-9, respectively. The modulation waveform is always a triangle wave and its shape is not programmable. An example of one period of the modulation waveform is shown in Figure 7-6.

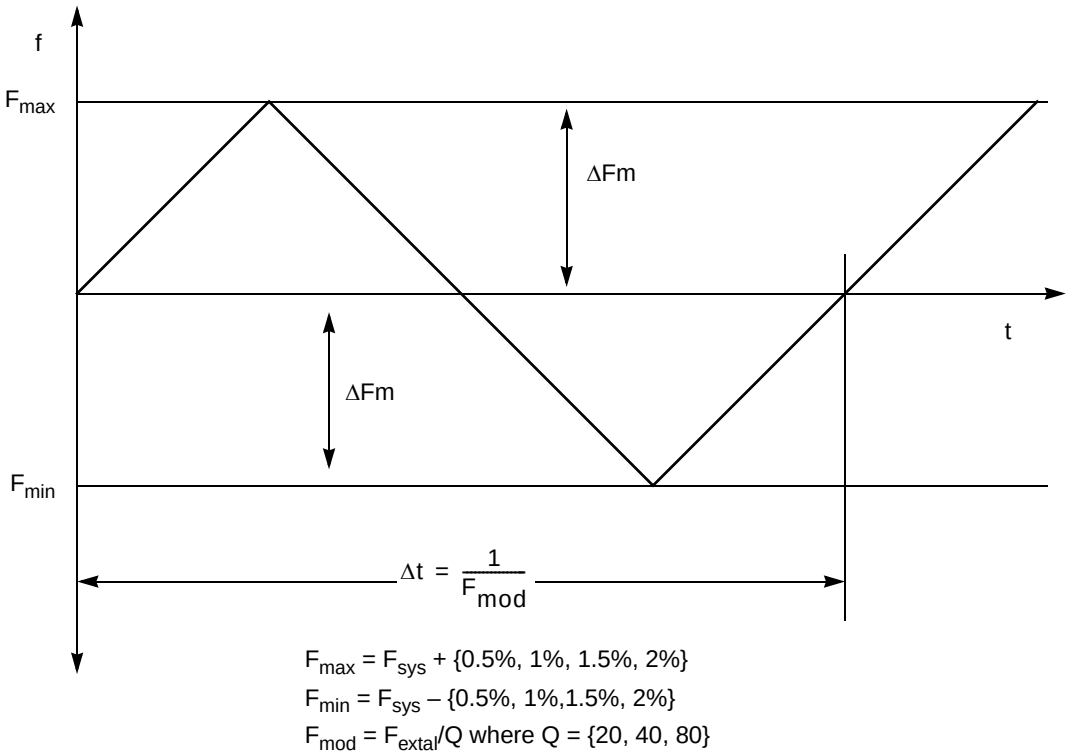


Figure 7-6. Frequency Modulation Waveform

7.4.3.4.1 Frequency Modulation Depth Calibration

The frequency modulation calibration system tunes a reference current into the modulation D/A so that the modulation depth (F_{max} and F_{min}) remains within specification. Disable frequency modulation prior to changing the EPREDIV, EMFD, or ERATE bit fields. Upon enabling frequency modulation a new calibration sequence is performed. A change to EPREDIV, EMFD, or ERATE while modulation is active invalidates calibration results.

This routine corrects for process variations, but because temperature can change after the calibration has been performed, variation due to temperature drift is not eliminated. This system is also voltage dependent, so if the supply changes after the sequence takes place, error incurred is not corrected. The calibration system reuses the two counters in the lock detect circuit, the reference and feedback counters. The reference counter remains clocked by the reference clock, but the feedback counter is clocked by the VCO clock.

When the calibration routine is initiated by writing to the EDEPTH bits, the CALPASS and CALDONE status bits are immediately cleared.

When calibration is induced the VCO is given time to settle before the feedback and reference counters start counting. Full VCO clock cycles are counted by the feedback counter during this time to give the initial center frequency count. When the reference counter has counted to the programmed number of reference count cycles, the input to the feedback counter is disabled and the result is placed in the COUNT0 register. The calibration system then enables modulation at programmed ΔFm and the VCO gets time to settle. Both counters are reset and restarted. The feedback counter begins to count full VCO clock

cycles again to obtain the delta-frequency count. The counter runs only during the high phase of the triangular modulation waveform. Several half-modulation periods are measured during the calibration routine to increase the resolution of the frequency measurement. This results in a measurement of the average frequency during the high phase of the modulation waveform, which under ideal circumstances is equivalent to one-half of the desired modulation depth. When the reference counter has counted to the new programmed number of reference count cycles, the feedback counter is stopped again.

The delta-frequency count minus the center frequency count (COUNT0) results in a delta count proportional to the reference current into the modulation D/A. That delta count is subtracted from the expected value for the selected depth resulting in an error count. The sign of this error count determines the direction taken by the calibration D/A to update the calibration current. After obtaining the error count for the present iteration, both counters are cleared. The stored count of COUNT0 is preserved while a new feedback count is obtained, and the process to determine the error count is repeated. The calibration system repeats this process eight times, once for each bit of the calibration D/A.

After the last decision is made, a 1 is written to the CALDONE bit of the SYNSR. If an error occurs during the calibration routine, then CALPASS remains 0. If the routine completed successfully, CALPASS is set to 1.

7.4.3.4.2 Programming System Clock Frequency With Frequency Modulation

The following steps illustrate proper programming of the frequency modulation mode. These steps ensure proper operation of the calibration routine and prevent frequency overshoot from the sequence. The PLL should be programmed and allowed to lock in non-FM mode at the desired frequency as outlined in Section 7.4.3.3.5, Programming System Clock Frequency.

1. Monitor LOCK bit. Do not proceed until the PLL is locked in non-modulation mode.
2. Write a value of ERFD = ERFD + 1 to the ERFD field of the ESYNCR2 to ensure the maximum system frequency is not exceeded during the calibration routine. This should have been done when allowing the PLL to lock in non-FM mode.
3. Program the desired modulation rate and depth to the ERATE and EDEPTH bitfields simultaneously using a single 32-bit write to the ESYNCR register2. Setting ERATE alone may set the LOLF flag. This action initiates the calibration sequence.
4. Allow time for the calibration sequence. Wait for the PLL to lock (i.e.,the LOCK bit to set in the SYNSR). At this time CALDONE should be asserted. CALPASS is asserted if the calibration was successful. If not, the calibration can be re-initiated by repeating from step 3. When the PLL achieves lock, write the ERFD value desired.

The frequency modulation system is dependent on several factors, including the accuracies of the V_{DDSYN}/V_{SSSYN} voltage, of the crystal oscillator frequency, and of the manufacturing variation.

For example, if a 5% accurate supply voltage is used, then a 5% modulation depth error results. If the crystal oscillator frequency is skewed from the nominal operating frequency, the resulting modulation frequency is proportionally skewed. Finally, the error due to the manufacturing and environment variation alone can cause the frequency modulation depth error to be greater than 20%.

7.5 Resets

This section describes the reset operation of the PLL, including power-on reset and normal resets. The reset values of registers and signals are provided in other sections.

7.5.1 Clock Mode Selection

The initial clock mode is reflected in the MODE, PLLSEL, and PLLREF bits of the synthesizer status register (SYNSR) as well as the ESYNCR1[CLKCFG] bit field. The clock mode can be modified by writing to the CLKCFG bit field. The synthesizer status register then reflects the newly-selected PLL clock mode.

Table 7-13 summarizes clock mode selection.

Table 7-13. Clock Mode Selection

Clock Mode	Synthesizer Status Register (SYNSR) MODE, PLLSEL, and PLLREF Bits		
	MODE/ CLKCFG2	PLLSEL/ CLKCFG1	PLLREF/ CLKCFG0
PLL Off mode	0	X	X
Normal mode with external reference	1	1	0
Normal mode with crystal reference	1	1	1
Reserved	1	0	0

7.5.1.1 Power-On Reset (POR)

The PLL will not operate until the POR signal has been deasserted and the ESYNCR1[CLKCFG] bitfield set for PLL mode. Refer to *PXN20 Microcontroller Data Sheet* for these thresholds. At this point, the PLL operates in self-clocked mode (SCM) until a valid reference clock is detected by the internal clock monitor circuit.

Internal to the PLL, the VCO is held in reset until the negation of the POR signal. This prevents the PLL from attempting to lock before its supplies are within specification, which can cause VCO/loop gain to be lower than what the analog loop is designed for.

7.5.1.2 External Reset

After POR has negated, the PLL defaults to PLL Off mode and the default clock source for the system clock is the 16 MHz_IRC. After reset exit, the PLL may be configured for operation and after lock may be selected as the system clock source.

After the initial lock with the default MFD (assuming normal mode was selected), ESYNCR1 may be written to modify the MFD for the desired operating frequency. The PLL may not lock with an MFD and crystal frequency combination that attempts to force the VCO outside of its operating range.

CAUTION

When running in an unlocked state, the clocks generated by the PLL are not guaranteed stable and may exceed the maximum specified operating frequency of the device. The RFD should always be used as described in Section 7.4.3.3.5, Programming System Clock Frequency, to insulate the system from any potential frequency overshoot of the PLL clocks.

7.5.2 PLL Loss-of-Lock Reset

By programming the LOLRE bit in the ESYNCR2, the PLL can assert reset when a loss-of-lock condition occurs. Because the LOCK and LOCKS bits in the SYNSR are re-initialized after reset, the SIU reset status register described in Chapter 8, System Integration Unit (SIU), must be read to determine a loss-of-lock condition occurred.

In PLL Off mode, the PLL cannot lock; therefore a loss-of-lock condition cannot occur and LOLRE has no effect.

7.5.3 PLL Loss-of-Clock Reset

When a loss-of-clock condition is recognized, $\overline{\text{RESET}}$ is asserted if the LOCRE bit in the SYNCR is set. The LOCF and LOC bits in the SYNSR are cleared after reset, therefore, the LOC bit must be read in the SIU_RSR to determine that a loss-of-clock condition occurred. LOCRE has no effect in PLL Off mode.

7.6 Interrupts

This section describes the interrupt requests that the PLL can generate.

7.6.1 Loss-of-Lock Interrupt Request

By setting the LOLIRQ bit in the ESYNCR2, the PLL can request an interrupt when a loss-of-lock condition occurs.

In PLL Off mode, the PLL cannot lock; therefore a loss-of-lock condition cannot occur and the LOLIRQ has no effect.

7.6.2 Loss-of-Clock Interrupt Request

When a loss-of-clock condition is recognized, the PLL requests an interrupt if the LOCIRQ bit in the SYNCR is set. The LOCIRQ bit has no effect in PLL Off mode or if LOCEN is equal to 0.

Chapter 8
System Integration Unit (SIU)

8.1 Introduction

The system integration unit (SIU) controls MCU reset configuration, the system reset operation, pad configuration, external interrupt, general purpose I/O (GPIO), internal peripheral multiplexing, clock frequency divider configuration, peripheral clock disable configuration, and peripheral clock disable acknowledge. The reset configuration block contains the external pin boot configuration logic. The pad configuration block controls the static electrical characteristics of I/O pins. The GPIO block provides uniform and discrete input/output control of the MCU I/O pins. The reset controller performs reset monitoring of internal and external reset sources, and drives the $\overline{\text{RESET}}$ pin. The core accesses the SIU through the peripheral bus.

8.1.1 Block Diagram

Figure 8-1 is a block diagram of the SIU. The signals shown are external pins to the device. The SIU registers are accessed through the crossbar switch. The power-on reset (POR) detection block, pad interface/pad ring block, and peripheral I/O channels are external to the SIU.

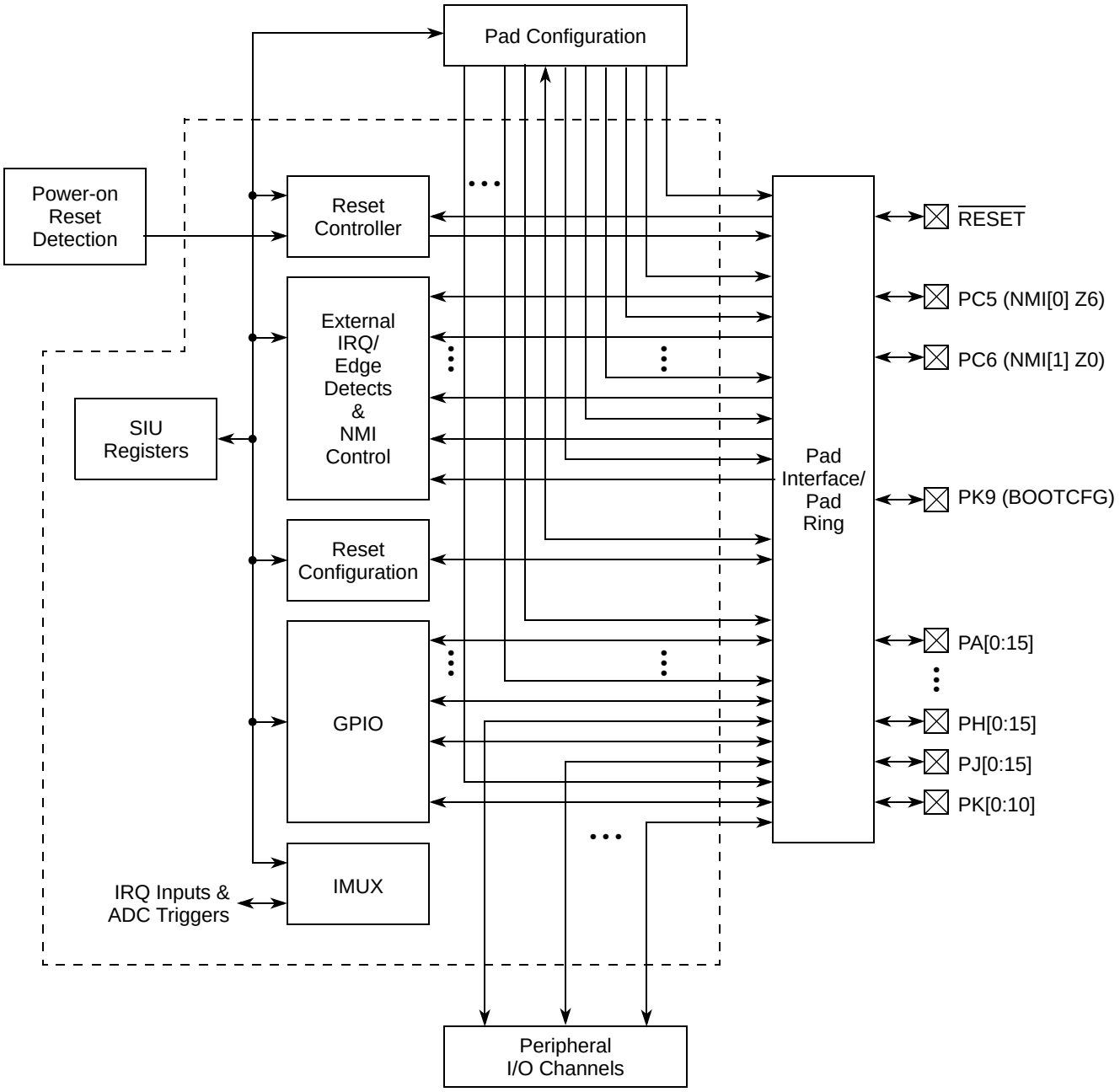


Figure 8-1. SIU Block Diagram

8.1.2 Features

Features of the SIU include the following:

- System configuration
  - MCU reset configuration via external pins
  - Pad configuration control
- System reset monitoring and generation

  - Power-on reset support
  - Reset status register providing last reset source to software
  - Software controlled reset assertion
- External interrupt
  - 16 interrupt requests (139 inputs multiplexed down to 16 inputs in eight groups of 16 and one group of 11, on Ports A through K)
  - Rising or falling edge event detection
  - Programmable digital filter for glitch rejection
- GPIO
  - GPIO function on as many as 155 I/O pins
  - Dedicated input and output registers for each GPIO pin.
  - Parallel input and output registers with pins grouped into 16-bit ports (Ports A through K)
    - Read/Write data is coherent with data written/read using dedicated input/output registers
- Internal multiplexing
  - Allows flexible selection of ADC trigger inputs
  - Allows selection of interrupt requests among external pins
  - Allows selection of eMIOS inputs between external pins and deserialized DSPI outputs
  - Allows selection of eMIOS outputs or SIU data register to be serialized via the DSPI
- System clock control
  - Clock divider control for individual peripherals or peripheral groups for lower power operation
  - Halt request registers to disable clocks to unused peripherals for lower power operation
  - Halt acknowledge registers to determine when peripheral clocks are disabled

8.1.3 Modes of Operation

8.1.3.1 Normal Mode

In normal mode, the SIU provides the register interface and logic that controls system configuration, the reset controller, GPIO, clock divider control, and peripheral clock disable/acknowledge.

8.1.3.2 Debug Mode

SIU operation in debug mode is identical to normal mode operation.

8.2 External Signal Description

Refer to Table 3-1 and Section 3.4, Detailed Signal Description, for signal properties.

8.2.1 Ports vs. General-Purpose I/O Pins

The PXN20 provides 155 individual GPIO pins, organized into 10 ports named Port A through Port K. Port I is omitted from the series of ports. Of these ports, Ports A through J provide 16 pins each, and Port K provides 11 pins.

The GPIO pins provide general-purpose input and output function. The GPIO pins are multiplexed with other I/O pin functions. A pad control register (SIU_PCR) sets the multiplexing and other functions for the pins. An input (SIU_GPDI) or output (SIU_GPDO) register controls each GPIO input and output separately. Pins correspond to numbered control registers beginning with Port A (pin A0) and continuing consecutively to Port K (pin K10). Registers are numbered consecutively from 0 to 154. See the following:

- Section 8.3.2.13, Pad Configuration Registers (SIU_PCR)
- Section 8.3.2.14, GPIO Pin Data Output Registers (SIU_GPDO16_19–SIU_GPDO152_154)
- Section 8.3.2.15, GPIO Pin Data Input Registers (SIU_GPDI0_3–SIU_GPDI152_154)
- Section 8.3.2.27, Parallel GPIO Pin Data Output Register 0 (SIU_PGPDO0) through Section 8.3.2.36, Parallel GPIO Pin Data Input Register 4 (SIU_PGPDI4)
- Section 8.3.2.37, Masked Parallel GPIO Pin Data Output Register 1 (SIU_MPGPDO1).

8.3 Memory Map and Registers

This section provides a detailed description of all DSPI registers.

8.3.1 Module Memory Map

Table 8-1 is the address map for the SIU registers. All register addresses are given as an offset of the SIU base address.

Table 8-1. SIU Memory Map

Offset from SIU_BASE (0xFFFE_8000)	Register	Access	Reset Value	Section/Page
0x0000–0x0003	Reserved			
0x0004	SIU_MIDR—MCU ID register	R	—[1]	8.3.2.1/8-13
0x0008–0x000B	Reserved			
0x000C	SIU_RSR—Reset status register	R	0x8000_000U	8.3.2.2/8-14
0x0010	SIU_SRCR—System reset control register	R/W	0x0800_C000	8.3.2.3/8-15
0x0014	SIU_EISR—SIU external interrupt status register	R/W	0x0000_0000	8.3.2.4/8-16
0x0018	SIU_DIRER—DMA/interrupt request enable register	R/W	0x0000_0000	8.3.2.5/8-17
0x001C	SIU_DIRSR—DMA/interrupt request select register	R/W	0x0000_0000	8.3.2.6/8-18
0x0020	SIU_OSR—Overrun status register	R/W	0x0000_0000	8.3.2.7/8-19
0x0024	SIU_ORER—Overrun request enable register	R/W	0x0000_0000	8.3.2.8/8-19
0x0028	SIU_IREER—External IRQ rising-edge event enable register	R/W	0x0000_0000	8.3.2.9/8-20

Table 8-1. SIU Memory Map (continued)

Offset from SIU_BASE (0xFFFE_8000)	Register	Access	Reset Value	Section/Page
0x002C	SIU_IFEER—External IRQ falling-edge event enable register	R/W	0x0000_0000	8.3.2.10/8-20
0x0030	SIU_IDFR—External IRQ digital filter register	R/W	0x0000_0000	8.3.2.11/8-21
0x0034	SIU_IFIR—External IRQ filtered input register	R/W	0x0000_0000	8.3.2.12/8-22
0x0038–0x003F	Reserved			
0x0040–0x0174	SIU_PCR0 – SIU_PCR154—Pad configuration register 0 – Pad configuration register 154	R/W	—[1]	8.3.2.13/8-22
0x0176–0x060F	Reserved			
0x0610–0x0689	SIU_GPDO16_19 – SIU_GPDO152_154—GPIO pin data output register 16-19–GPIO pin data output register 152–154	R/W	0x0000_0000	8.3.2.14/8-26
0x0690–0x07FF	Reserved			
0x0800–0x0898	SIU_GPDI0_3 – SIU_GPDI152_155—GPIO pin data input register 0–3 –GPIO pin data input register 152–154	R/W	—[1]	8.3.2.15/8-28
0x089C–0x0903	Reserved			
0x0904	SIU_ISEL1—External interrupt select register 1	R/W	0x0000_0000	8.3.2.16/8-29
0x0908	SIU_ISEL2—External interrupt select register 2	R/W	0x0000_0000	8.3.2.17/8-33
0x090C–0x090F	Reserved			
0x0910	SIU_ISEL4—ADC trigger input select register 4	R/W	0x0000_0000	8.3.2.18/8-35
0x0914–0x097F	Reserved			
0x0980	SIU_CCR—Chip configuration register	R/W	0x000U_0000	8.3.2.19/8-36
0x0984	SIU_ECCR—External clock control register	R/W	0x0000_1001	8.3.2.20/8-37
0x0988	SIU_GPR0—General purpose register 0	R/W	0x0000_0000	8.3.2.21/8-38
0x098C	SIU_GPR1—General purpose register 1	R/W	0x0000_0000	8.3.2.21/8-38
0x0990	SIU_GPR2—General purpose register 2	R/W	0x0000_0000	8.3.2.21/8-38
0x0994	SIU_GPR3—General purpose register 3	R/W	0x0000_0000	8.3.2.21/8-38
0x0998–0x099F	Reserved			
0x09A0	SIU_SYSCLK—System clock register	R/W	0x0000_0000	8.3.2.22/8-38
0x09A4	SIU_HLT0—Halt request register 0	R/W	0x0000_0000	8.3.2.23/8-39
0x09A8	SIU_HLT1—Halt request register 1	R/W	0x0000_0000	8.3.2.23/8-39
0x09AC	SIU_HLTACK0—Halt acknowledge register 0	R	0x0000_0000	8.3.2.24/8-41
0x09B0	SIU_HLTACK1—Halt acknowledge register 1	R	0x0000_0000	8.3.2.24/8-41
0x09B4	SIU_EMIOS_SEL0—eMIOS select register 0	R/W	0x0000_0000	8.3.2.25/8-44
0x09B8	SIU_EMIOS_SEL1—eMIOS select register 1	R/W	0x0000_0000	8.3.2.25/8-44
0x09BC	SIU_EMIOS_SEL2—eMIOS select register 2	R/W	0x0000_0000	8.3.2.25/8-44

Table 8-1. SIU Memory Map (continued)

Offset from SIU_BASE (0xFFFE_8000)	Register	Access	Reset Value	Section/Page
0x09C0	SIU_EMIOS_SEL3—eMIOS select register 3	R/W	0x0000_0000	8.3.2.25/8-44
0x09C4	SIU_ISEL2A—External interrupt select register 2A	R/W	0x0000_0000	8.3.2.26/8-45
0x09C8–0x0BFF	Reserved			
0x0C00	SIU_PGPDO0—Parallel GPIO pin data output register 0	R/W	0x0000_0000	8.3.2.27/8-48
0x0C04	SIU_PGPDO1—Parallel GPIO pin data output register 1	R/W	0x0000_0000	8.3.2.28/8-48
0x0C08	SIU_PGPDO2—Parallel GPIO pin data output register 2	R/W	0x0000_0000	8.3.2.29/8-49
0x0C0C	SIU_PGPDO3—Parallel GPIO pin data output register 3	R/W	0x0000_0000	8.3.2.30/8-49
0x0C10	SIU_PGPDO4—Parallel GPIO pin data output register 4	R/W	0x0000_0000	8.3.2.31/8-49
0x0C14–0x0C3F	Reserved			
0x0C40	SIU_PGPDI0—Parallel GPIO pin data input register 0	R	—[1]	8.3.2.32/8-50
0x0C44	SIU_PGPDI1—Parallel GPIO pin data input register 1	R	—[1]	8.3.2.33/8-50
0x0C48	SIU_PGPDI2—Parallel GPIO pin data input register 2	R	—[1]	8.3.2.34/8-51
0x0C4C	SIU_PGPDI3—Parallel GPIO pin data input register 3	R	—[1]	8.3.2.35/8-51
0x0C50	SIU_PGPDI4—Parallel GPIO pin data input register 4	R	—[1]	8.3.2.36/8-52
0x0C54–0x0C83	Reserved			
0x0C84	SIU_MPGPDO1—Masked parallel GPIO data output register 1	W	0x0000_0000	8.3.2.37/8-52
0x0C88	SIU_MPGPDO2—Masked parallel GPIO data output register 2	W	0x0000_0000	8.3.2.38/8-53
0x0C8C	SIU_MPGPDO3—Masked parallel GPIO data output register 3	W	0x0000_0000	8.3.2.39/8-53
0x0C90	SIU_MPGPDO4—Masked parallel GPIO data output register 4	W	0x0000_0000	8.3.2.40/8-54
0x0C94	SIU_MPGPDO5—Masked parallel GPIO data output register 5	W	0x0000_0000	8.3.2.41/8-54
0x0C98	SIU_MPGPDO6—Masked parallel GPIO data output register 6	W	0x0000_0000	8.3.2.42/8-55
0x0C9C	SIU_MPGPDO7—Masked parallel GPIO data output register 7	W	0x0000_0000	8.3.2.43/8-55
0x0CA0	SIU_MPGPDO8—Masked parallel GPIO data output register 8	W	0x0000_0000	8.3.2.44/8-56
0x0CA4	SIU_MPGPDO9—Masked parallel GPIO data output register 9	W	0x0000_0000	8.3.2.45/8-56
0x0CA8–0x0CFF	Reserved			
0x0D00	SIU_DSPIAH—Masked serial GPO register for DSPI_A high	R/W	0x0000_0000	8.3.2.46/8-57
0x0D04	SIU_DSPIAL—Masked serial GPO register for DSPI_A low	R/W	0x0000_0000	8.3.2.47/8-58
0x0D08	SIU_DSPIBH—Masked serial GPO register for DSPI_B high	R/W	0x0000_0000	8.3.2.48/8-58
0x0D0C	SIU_DSPIBL—Masked serial GPO register for DSPI_B low	R/W	0x0000_0000	8.3.2.49/8-59
0x0D10	SIU_DSPICH—Masked serial GPO register for DSPI_C high	R/W	0x0000_0000	8.3.2.50/8-60
0x0D14	SIU_DSPICL—Masked serial GPO register for DSPI_C low	R/W	0x0000_0000	8.3.2.51/8-60

Table 8-1. SIU Memory Map (continued)

Offset from SIU_BASE (0xFFFE_8000)	Register	Access	Reset Value	Section/Page
0x0D18	SIU_DSPIDH—Masked serial GPO register for DSPI_D high	R/W	0x0000_0000	8.3.2.52/8-61
0x0D1C	SIU_DSPIDL—Masked serial GPO register for DSPI_D low	R/W	0x0000_0000	8.3.2.53/8-62
0x0D20–0x0D43	Reserved			
0x0D44	SIU_EMIOSA—eMIOS select register for DSPI_A	R/W	0x0000_0000	8.3.2.54/8-62
0x0D48	SIU_DSPIAHLA—SIU_DSPIAH/L select register for DSPI_A	R/W	0x0000_0000	8.3.2.55/8-63
0x0D4C–0x0D53	Reserved			
0x0D54	SIU_EMIOSB—eMIOS select register for DSPI_B	R/W	0x0000_0000	8.3.2.56/8-64
0x0D58	SIU_DSPIBHLB—SIU_DSPIBH/L select register for DSPI_B	R/W	0x0000_0000	8.3.2.57/8-64
0x0D5C–0x0D63	Reserved			
0x0D64	SIU_EMIOSC—eMIOS select register for DSPI_C	R/W	0x0000_0000	8.3.2.58/8-65
0x0D68	SIU_DSPICHLC—H/L select register for DSPI_C	R/W	0x0000_0000	8.3.2.59/8-65
0x0D6C–0x0D73	Reserved			
0x0D74	SIU_EMIOSD—eMIOS select register for DSPI_D	R/W	0x0000_0000	8.3.2.60/8-66
0x0D78–0x0D7B	SIU_DSPIDHLD—SIU_DSPIDH/L select register for DSPI_D	R/W	0x0000_0000	8.3.2.61/8-67
0x0D7C–0x3FFF	Reserved			

[1] See register description for reset value.

Table 8-2 provides absolute hex addresses for the SIU_PCR and SIU_GPDO registers.

Table 8-2. Detailed Memory Map for SIU_PCR, SIU_GPDO, and SIU_GPDI

Pad ID	Pad #	SIU_PCR Address	SIU_GPDO Address	SIU_GPDI Address
PA0	0	FFFE_8040	PA[15:0] are inputs only	FFFE_8800
PA1	1	FFFE_8042		FFFE_8801
PA2	2	FFFE_8044		FFFE_8802
PA3	3	FFFE_8046		FFFE_8803
PA4	4	FFFE_8048		FFFE_8804
PA5	5	FFFE_804A		FFFE_8805
PA6	6	FFFE_804C		FFFE_8806
PA7	7	FFFE_804E		FFFE_8807
PA8	8	FFFE_8050		FFFE_8808
PA9	9	FFFE_8052		FFFE_8809
PA10	10	FFFE_8054		FFFE_880A
PA11	11	FFFE_8056		FFFE_880B
PA12	12	FFFE_8058		FFFE_880C
PA13	13	FFFE_805A		FFFE_880D
PA14	14	FFFE_805C		FFFE_880E
PA15	15	FFFE_805E		FFFE_880F
PB0	16	FFFE_8060	FFFE_8610	FFFE_8810
PB1	17	FFFE_8062	FFFE_8611	FFFE_8811
PB2	18	FFFE_8064	FFFE_8612	FFFE_8812
PB3	19	FFFE_8066	FFFE_8613	FFFE_8813
PB4	20	FFFE_8068	FFFE_8614	FFFE_8814
PB5	21	FFFE_806A	FFFE_8615	FFFE_8815
PB6	22	FFFE_806C	FFFE_8616	FFFE_8816
PB7	23	FFFE_806E	FFFE_8617	FFFE_8817
PB8	24	FFFE_8070	FFFE_8618	FFFE_8818
PB9	25	FFFE_8072	FFFE_8619	FFFE_8819
PB10	26	FFFE_8074	FFFE_861A	FFFE_881A
PB11	27	FFFE_8076	FFFE_861B	FFFE_881B
PB12	28	FFFE_8078	FFFE_861C	FFFE_881C
PB13	29	FFFE_807A	FFFE_861D	FFFE_881D
PB14	30	FFFE_807C	FFFE_861E	FFFE_881E
PB15	31	FFFE_807E	FFFE_861F	FFFE_881F

Table 8-2. Detailed Memory Map for SIU_PCR, SIU_GPDO, and SIU_GPDI (continued)

Pad ID	Pad #	SIU_PCR Address	SIU_GPDO Address	SIU_GPDI Address
PC0	32	FFFE_8080	FFFE_8620	FFFE_8820
PC1	33	FFFE_8082	FFFE_8621	FFFE_8821
PC2	34	FFFE_8084	FFFE_8622	FFFE_8822
PC3	35	FFFE_8086	FFFE_8623	FFFE_8823
PC4	36	FFFE_8088	FFFE_8624	FFFE_8824
PC5	37	FFFE_808A	FFFE_8625	FFFE_8825
PC6	38	FFFE_808C	FFFE_8626	FFFE_8826
PC7	39	FFFE_808E	FFFE_8627	FFFE_8827
PC8	40	FFFE_8090	FFFE_8628	FFFE_8828
PC9	41	FFFE_8092	FFFE_8629	FFFE_8829
PC10	42	FFFE_8094	FFFE_862A	FFFE_882A
PC11	43	FFFE_8096	FFFE_862B	FFFE_882B
PC12	44	FFFE_8098	FFFE_862C	FFFE_882C
PC13	45	FFFE_809A	FFFE_862D	FFFE_882D
PC14	46	FFFE_809C	FFFE_862E	FFFE_882E
PC15	47	FFFE_809E	FFFE_862F	FFFE_882F
PD0	48	FFFE_80A0	FFFE_8630	FFFE_8830
PD1	49	FFFE_80A2	FFFE_8631	FFFE_8831
PD2	50	FFFE_80A4	FFFE_8632	FFFE_8832
PD3	51	FFFE_80A6	FFFE_8633	FFFE_8833
PD4	52	FFFE_80A8	FFFE_8634	FFFE_8834
PD5	53	FFFE_80AA	FFFE_8635	FFFE_8835
PD6	54	FFFE_80AC	FFFE_8636	FFFE_8836
PD7	55	FFFE_80AE	FFFE_8637	FFFE_8837
PD8	56	FFFE_80B0	FFFE_8638	FFFE_8838
PD9	57	FFFE_80B2	FFFE_8639	FFFE_8839
PD10	58	FFFE_80B4	FFFE_863A	FFFE_883A
PD11	59	FFFE_80B6	FFFE_863B	FFFE_883B
PD12	60	FFFE_80B8	FFFE_863C	FFFE_883C
PD13	61	FFFE_80BA	FFFE_863D	FFFE_883D
PD14	62	FFFE_80BC	FFFE_863E	FFFE_883E
PD15	63	FFFE_80BE	FFFE_863F	FFFE_883F

Table 8-2. Detailed Memory Map for SIU_PCR, SIU_GPDO, and SIU_GPDI (continued)

Pad ID	Pad #	SIU_PCR Address	SIU_GPDO Address	SIU_GPDI Address
PE0	64	FFFE_80C0	FFFE_8640	FFFE_8840
PE1	65	FFFE_80C2	FFFE_8641	FFFE_8841
PE2	66	FFFE_80C4	FFFE_8642	FFFE_8842
PE3	67	FFFE_80C6	FFFE_8643	FFFE_8843
PE4	68	FFFE_80C8	FFFE_8644	FFFE_8844
PE5	69	FFFE_80CA	FFFE_8645	FFFE_8845
PE6	70	FFFE_80CC	FFFE_8646	FFFE_8846
PE7	71	FFFE_80CE	FFFE_8647	FFFE_8847
PE8	72	FFFE_80D0	FFFE_8648	FFFE_8848
PE9	73	FFFE_80D2	FFFE_8649	FFFE_8849
PE10	74	FFFE_80D4	FFFE_864A	FFFE_884A
PE11	75	FFFE_80D6	FFFE_864B	FFFE_884B
PE12	76	FFFE_80D8	FFFE_864C	FFFE_884C
PE13	77	FFFE_80DA	FFFE_864D	FFFE_884D
PE14	78	FFFE_80DC	FFFE_864E	FFFE_884E
PE15	79	FFFE_80DE	FFFE_864F	FFFE_884F
PF0	80	FFFE_80E0	FFFE_8650	FFFE_8850
PF1	81	FFFE_80E2	FFFE_8651	FFFE_8851
PF2	82	FFFE_80E4	FFFE_8652	FFFE_8852
PF3	83	FFFE_80E6	FFFE_8653	FFFE_8853
PF4	84	FFFE_80E8	FFFE_8654	FFFE_8854
PF5	85	FFFE_80EA	FFFE_8655	FFFE_8855
PF6	86	FFFE_80EC	FFFE_8656	FFFE_8856
PF7	87	FFFE_80EE	FFFE_8657	FFFE_8857
PF8	88	FFFE_80F0	FFFE_8658	FFFE_8858
PF9	89	FFFE_80F2	FFFE_8659	FFFE_8859
PF10	90	FFFE_80F4	FFFE_865A	FFFE_885A
PF11	91	FFFE_80F6	FFFE_865B	FFFE_885B
PF12	92	FFFE_80F8	FFFE_865C	FFFE_885C
PF13	93	FFFE_80FA	FFFE_865D	FFFE_885D
PF14	94	FFFE_80FC	FFFE_865E	FFFE_885E
PF15	95	FFFE_80FE	FFFE_865F	FFFE_885F

Table 8-2. Detailed Memory Map for SIU_PCR, SIU_GPDO, and SIU_GPDI (continued)

Pad ID	Pad #	SIU_PCR Address	SIU_GPDO Address	SIU_GPDI Address
PG0	96	FFFE_8100	FFFE_8660	FFFE_8860
PG1	97	FFFE_8102	FFFE_8661	FFFE_8861
PG2	98	FFFE_8104	FFFE_8662	FFFE_8862
PG3	99	FFFE_8106	FFFE_8663	FFFE_8863
PG4	100	FFFE_8108	FFFE_8664	FFFE_8864
PG5	101	FFFE_810A	FFFE_8665	FFFE_8865
PG6	102	FFFE_810C	FFFE_8666	FFFE_8866
PG7	103	FFFE_810E	FFFE_8667	FFFE_8867
PG8	104	FFFE_8110	FFFE_8668	FFFE_8868
PG9	105	FFFE_8112	FFFE_8669	FFFE_8869
PG10	106	FFFE_8114	FFFE_866A	FFFE_886A
PG11	107	FFFE_8116	FFFE_866B	FFFE_886B
PG12	108	FFFE_8118	FFFE_866C	FFFE_886C
PG13	109	FFFE_811A	FFFE_866D	FFFE_886D
PG14	110	FFFE_811C	FFFE_866E	FFFE_886E
PG15	111	FFFE_811E	FFFE_866F	FFFE_886F
PH0	112	FFFE_8120	FFFE_8670	FFFE_8870
PH1	113	FFFE_8122	FFFE_8671	FFFE_8871
PH2	114	FFFE_8124	FFFE_8672	FFFE_8872
PH3	115	FFFE_8126	FFFE_8673	FFFE_8873
PH4	116	FFFE_8128	FFFE_8674	FFFE_8874
PH5	117	FFFE_812A	FFFE_8675	FFFE_8875
PH6	118	FFFE_812C	FFFE_8676	FFFE_8876
PH7	119	FFFE_812E	FFFE_8677	FFFE_8877
PH8	120	FFFE_8130	FFFE_8678	FFFE_8878
PH9	121	FFFE_8132	FFFE_8679	FFFE_8879
PH10	122	FFFE_8134	FFFE_867A	FFFE_887A
PH11	123	FFFE_8136	FFFE_867B	FFFE_887B
PH12	124	FFFE_8138	FFFE_867C	FFFE_887C
PH13	125	FFFE_813A	FFFE_867D	FFFE_887D
PH14	126	FFFE_813C	FFFE_867E	FFFE_887E
PH15	127	FFFE_813E	FFFE_867F	FFFE_887F

Table 8-2. Detailed Memory Map for SIU_PCR, SIU_GPDO, and SIU_GPDI (continued)

Pad ID	Pad #	SIU_PCR Address	SIU_GPDO Address	SIU_GPDI Address
PJ0	128	FFFE_8140	FFFE_8680	FFFE_8880
PJ1	129	FFFE_8142	FFFE_8681	FFFE_8881
PJ2	130	FFFE_8144	FFFE_8682	FFFE_8882
PJ3	131	FFFE_8146	FFFE_8683	FFFE_8883
PJ4	132	FFFE_8148	FFFE_8684	FFFE_8884
PJ5	133	FFFE_814A	FFFE_8685	FFFE_8885
PJ6	134	FFFE_814C	FFFE_8686	FFFE_8886
PJ7	135	FFFE_814E	FFFE_8687	FFFE_8887
PJ8	136	FFFE_8150	FFFE_8688	FFFE_8888
PJ9	137	FFFE_8152	FFFE_8689	FFFE_8889
PJ10	138	FFFE_8154	FFFE_868A	FFFE_888A
PJ11	139	FFFE_8156	FFFE_868B	FFFE_888B
PJ12	140	FFFE_8158	FFFE_868C	FFFE_888C
PJ13	141	FFFE_815A	FFFE_868D	FFFE_888D
PJ14	142	FFFE_815C	FFFE_868E	FFFE_888E
PJ15	143	FFFE_815E	FFFE_868F	FFFE_888F
PK0	144	FFFE_8160	FFFE_8690	FFFE_8890
PK1	145	FFFE_8162	FFFE_8691	FFFE_8891
PK2	146	FFFE_8164	FFFE_8692	FFFE_8892
PK3	147	FFFE_8166	FFFE_8693	FFFE_8893
PK4	148	FFFE_8168	FFFE_8694	FFFE_8894
PK5	149	FFFE_816A	FFFE_8695	FFFE_8895
PK6	150	FFFE_816C	FFFE_8696	FFFE_8896
PK7	151	FFFE_816E	FFFE_8697	FFFE_8897
PK8	152	FFFE_8170	FFFE_8698	FFFE_8898
PK9	153	FFFE_8172	FFFE_8699	FFFE_8899
PK10	154	FFFE_8174	FFFE_869A	FFFE_889A

8.3.2 Register Descriptions

This section lists the DSPI registers in address order and describes the registers and their bit fields.

8.3.2.1 MCU ID Register (SIU_MIDR)

The SIU_MIDR contains the part identification number, package type, and mask revision number specific to the device. The part number is a read-only field mask-programmed with the device part number. It is not changed for bug fixes or process changes. The package type is a read-only field that reflects the device package type. The mask number is a read-only field mask-programmed with the device's specific mask revision level.

Offset: SIU_BASE + 0x0004 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	PARTNUM															
W																
Reset	0	1	0	1	0	1	1	0	0	1	1	0	1	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CSP	PKG					0	0	MASKNUM_MAJOR				MASKNUM_MINOR			
W																
Reset[1]	0	*	*	*	*	*	0	0	*	*	*	*	*	*	*	*

Figure 8-2. MCU ID Register (SIU_MIDR)

[1] PKG default value reflects the device package type as defined in Table 8-3.
MASKNUM_MAJOR default value is 0x0 for the device's initial mask set and changes for each major mask set revision.
MASKNUM_MINOR default value is 0x0 for the device's initial mask set and changes for each minor mask set revision.

Table 8-3. SIU_MIDR Field Descriptions

Field	Description
PARTNUM	MCU Part Number. Read-only, mask-programmed part identification number of the MCU. Reads 0x5668 for the PXN20.
CSP	Chip Scale Package. The CSP bit indicates whether the die is mounted in a chip scale package. 0 Not a chip scale package. 1 Chip scale package.
PKG	Package Configuration. These values set the pin package used for each PXN20 device. 0b10000 208-pin MAPBGA 0b00000 256-pin MAPBGA All other combinations are reserved.
MASKNUM_MAJOR	Major Mask Revision Number. Read-only, mask-programmed mask number of the MCU. Reads 0x0 for the device's initial mask set and changes for each major mask set revision.
MASKNUM_MINOR	Minor Mask Revision Number. Read-only, mask-programmed mask number of the MCU. Reads 0x0 for the device's initial mask set and changes for each minor mask set.

8.3.2.2 Reset Status Register (SIU_RSR)

The SIU_RSR reflects the most recent source, or reset sources, and the pins' configuration state at reset. This register contains one bit for each reset source, indicating the last reset was power-on reset (POR), external, software system, watchdog, loss of PLL lock, loss of clock, or checkstop reset. A reset status bit set to logic one indicates the reset type that occurred. After it is set, the reset source status bits in the SIU_RSR remain set until another reset occurs. In the following cases more than one reset bit is set:

1. If any reset request has negated and the device is still in the resulting reset, and then an external reset is requested, both the original reset type and external reset status bits are set. In this case, the device started the reset sequence due to due to a non-external reset request but ended the reset sequence after an external reset request.
2. If any of the loss of clock, loss of lock, watchdog or checkstop reset requests occur on the same clock cycle, and no other higher priority reset source is requesting reset (Table 8-4), the reset status bits for all of the requesting resets are set.

Simultaneous reset requests are prioritized. When reset requests of different priorities occur on the same clock cycle, the lower priority reset request is ignored. Only the highest priority reset request's status bit is set. Except for a power-on reset request and condition 1 above, all reset requests of any priority are ignored until the device exits reset.

Table 8-4. Reset Source Priorities

Reset Source	Priority
Power on reset (POR), LVI resets, and external reset (Group 0)	Highest
Software system reset (Group1)	.
Loss of clock, loss of lock, watchdog, checkstop (Group2)	.
Software external reset (Group 3)	Lowest

Offset: SIU_BASE + 0x000C — Access: User read only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	PORS	ERS	LLRS	LCRS	WDRS	CRS	0	0	0	0	0	0	0	0	SSRS	0
W																
Reset[1]	1[2]	0[3]	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	BOOT CFG	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	U[4]	0

1 The reset status register receives its reset values during power-on reset.

2 The PORS bit is also set on LVI or recovery from low-power sleep mode.

3 The ERS bit is also set if the RESET pin is held low to extend the reset sequence.

4 Before the rising edge of $\overline{\text{RESET}}$, the PK9 pin state sets the BOOTCFG bit value. During sleep mode recovery, this bit takes the state of PK9 when internal reset is negated.

Figure 8-3. Reset Status Register (SIU_RSR)

Table 8-5. SIU_RSR Field Descriptions

Field	Description
PORS	Power-on Reset Status. Set for any power-on or LVI reset. Also set on recovery from sleep mode. 0 The reset controller acknowledged another reset source since the last assertion of the power-on reset input. 1 The power-on reset input to the reset controller is asserted, and no other reset source has been acknowledged since that assertion of the power-on reset input except an external reset.
ERS	External Reset Status. (Asynchronous reset source) 0 Last reset source the reset controller acknowledged was not a valid assertion of the $\overline{\text{RESET}}$ pin. 1 Last reset source the reset controller acknowledged was a valid assertion of the $\overline{\text{RESET}}$ pin.
LLRS	Loss-of-Lock Reset Status. (Asynchronous reset source) 0 Last reset source the reset controller acknowledged was not a loss of PLL lock reset. 1 Last reset source the reset controller acknowledged was a loss of PLL lock reset.
LCRS	Loss-of-Clock Reset Status. (Asynchronous reset source) 0 Last reset source the reset controller acknowledged was not a loss of clock reset. 1 Last reset source the reset controller acknowledged was a loss of clock reset.
WDRS	Watchdog Timer Reset Status. 0 Last reset source the reset controller acknowledged was not a watchdog timer reset. 1 Last reset source the reset controller acknowledged was a watchdog timer reset.
CRS	Checkstop Reset Status. Set for Z6 or Z0 core reset. 0 Last reset source the reset controller acknowledged was not an enabled checkstop reset. 1 Last reset source the reset controller acknowledged was an enabled checkstop reset.
SSRS	Software System Reset Status. 0 Last reset source the reset controller acknowledged was not a software system reset. 1 Last reset source the reset controller acknowledged was a software system reset.
BOOTCFG	Status of BOOTCFG pin at negation of $\overline{\text{RESET}}$.

8.3.2.3 System Reset Control Register (SIU_SRCR)

Offset: SIU_BASE + 0x0010 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	SSR[1]	0	0	0	RSVD	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CRE1	CRE0	0	0	0	0	0	0	SSRL[3]	0	0	0	RSVD	0	0	0
W																
Reset	1[2]	1[2]	0	0	0	0	0	0	0	0	0	0	0	0	0	0

1 The SSR bit always reads as zero. A write of zero to this bit has no effect.

2 The CRE0/1 bits are reset to 0b1 by POR. Other resets sources do not reset the bit value.

3 Once written to a 1, the SSRL bit can be reset only to zero by POR.

Figure 8-4. System Reset Control Register (SIU_SRCR)

Table 8-6. SIU_SRCR Field Descriptions

Field	Description
SSR	Software System Reset. Used to generate a software system reset. Writing a 1 to this bit causes an internal reset. The software system reset is processed as a synchronous reset. The bit is automatically cleared on the assertion of any other reset source except a software external reset. 0 Do not generate a software system reset. 1 Generate a software system reset.
RSVD	Reserved for system use. Do not write to this bit.
CRE1	Checkstop Reset Enable (enable secondary CPU, Z0, checkstop to generate reset). Writing a 1 to this bit enables a reset when the e200z0 checkstop reset request input is asserted. The checkstop reset request input is a synchronous internal reset source. The CRE1 bit defaults to checkstop reset enabled at POR. If this bit is cleared, it remains cleared until the next POR. 0 No reset occurs when the e200z0 checkstop reset input to the reset controller is asserted. 1 A reset occurs when the e200z0 checkstop reset input to the reset controller is asserted.
CRE0	Checkstop Reset Enable (enable primary CPU, Z6, checkstop to generate reset). Writing a 1 to this bit enables a reset when the e200z6 checkstop reset request input is asserted. The checkstop reset request input is a synchronous internal reset source. The CRE0 bit defaults to checkstop reset enabled at POR. If this bit is cleared, it remains cleared until the next POR. 0 No reset occurs when the e200z6 checkstop reset input to the reset controller is asserted. 1 A reset occurs when the e200z6 checkstop reset input to the reset controller is asserted.
SSRL	Software System Reset Lock. This bit is used to disable the software system reset. This bit defaults to 0. A write of 1 disables the SSR bit until the next POR (write once). 0 Enable the SSR bit. 1 Disable the SSR bit.
RSVD	Reserved for system use. Do not write to this bit. **Note:** Once written to a 1, this bit can be reset to 0 only by POR. When this bit is set, bit 4 is automatically cleared.

8.3.2.4 External Interrupt Status Register (SIU_EISR)

The external interrupt status register is used to record edge-triggered events on the IRQ0–IRQ15 and NMI0–NMI1 inputs to the SIU. When an edge-triggered event is enabled in the SIU_IREER or SIU_IFEER for an IRQ*n* input and then sensed, the corresponding SIU_EISR flag bit is set. The IRQ flag bit is set, regardless of the state of the corresponding DMA/IRQ enable bit in SIU_DIRER (SIU_DIRER only affects interrupts and has no effect on enabling/selecting DMA requests). The IRQ flag bit remains set until cleared by software or through the servicing of a DMA request. The IRQ flag bits are cleared by writing a 1 to the bits. A write of 0 has no effect.

Offset: SIU_BASE + 0x0014 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	NMI0	NMI1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	w1c	w1c	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	EIF 15	EIF 14	EIF 13	EIF 12	EIF 11	EIF 10	EIF 9	EIF 8	EIF 7	EIF 6	EIF 5	EIF 4	EIF 3	EIF 2	EIF 1	EIF 0
W																
Reset	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c

Figure 8-5. SIU External Interrupt Status Register (SIU_EISR)

Table 8-7. SIU_EISR Field Descriptions

Field	Description
NMI0	Non-Maskable Interrupt Flag for primary CPU (Z6). NMI0 is for the primary core. This bit is set when an edge-triggered event occurs on the corresponding NMI0 input. 0 No edge-triggered event occurred on the corresponding NMI0 input. 1 An edge-triggered event occurred on the corresponding NMI0 input.
NMI1	Non-Maskable Interrupt Flag for secondary CPU (Z0). NMI1 is for the secondary core. This bit is set when an edge-triggered event occurs on the corresponding NMI1 input. 0 No edge-triggered event occurred on the corresponding NMI1 input. 1 An edge-triggered event occurred on the corresponding NMI1 input.
EIF*n*	External Interrupt Request Flag *n.* Set when an edge-triggered event occurs on the corresponding IRQ*n* input. 0 No edge triggered event occurred on the corresponding IRQ*n* input. 1 An edge triggered event occurred on the corresponding IRQ*n* input.

8.3.2.5 DMA/Interrupt Request Enable Register (SIU_DIRER)

The SIU_DIRER allows the assertion of a DMA or interrupt request if the corresponding flag bit is set in the SIU_EISR. The external interrupt request enable bits enable the interrupt. SIU_DIRER only affects interrupts and has no effect on enabling/selecting DMA requests. There are five interrupt requests from the SIU to the interrupt controller. The first four interrupts (0 to 3) go from the SIU to the interrupt controller. The remaining interrupts (4 to 15) are ORed together to form one additional source to the interrupt controller.

The EIRE bits allow selection of which external interrupt request flag bits cause assertion of the one interrupt request signal.

Offset: SIU_BASE + 0x0018 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	EIRE 15	EIRE 14	EIRE 13	EIRE 12	EIRE 11	EIRE 10	EIRE 9	EIRE 8	EIRE 7	EIRE 6	EIRE 5	EIRE 4	EIRE 3	EIRE 2	EIRE 1	EIRE 0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 8-6. SIU DMA/Interrupt Request Enable Register (SIU_DIRER)

Table 8-8. SIU_DIRER Field Descriptions

Field	Description
EIRE*n*	External Interrupt Request Enable *n.* Enables assertion of the interrupt request from the SIU to the interrupt controller when an edge triggered event occurs on the IRQ*n* pin. 0 External interrupt request disabled. 1 External interrupt request enabled.

8.3.2.6 DMA/Interrupt Request Select Register (SIU_DIRSR)

The SIU_DIRSR allows selection between a DMA or interrupt request for events on the IRQ1–IRQ0 inputs. The SIU_DIRSR selects between DMA and interrupt requests. If the corresponding bits are set in SIU_EISR and the SIU_DIRER, then the DMA/interrupt request select bit determines whether a DMA or interrupt request is asserted.

Offset: SIU_BASE + 0x001C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	DIRS1	DIRS0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 8-7. DMA/Interrupt Request Select Register (SIU_DIRSR)

Table 8-9. SIU_DIRSR Field Descriptions

Field	Description
DIRS*n*	DMA/Interrupt Request Select *n.* Selects between a DMA or interrupt request when an edge triggered event occurs on the corresponding IRQ*n* pin. 0 Interrupt request selected. 1 DMA request selected.

8.3.2.7 Overrun Status Register (SIU_OSR)

The SIU_OSR contains flag bits that record an overrun. These flag bits are cleared by writing 1 to the bits (w1c); writing 0 has no effect.

Offset: SIU_BASE + 0x0020 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	OVF 15	OVF 14	OVF 13	OVF 12	OVF 11	OVF 10	OVF 9	OVF 8	OVF 7	OVF 6	OVF 5	OVF 4	OVF 3	OVF 2	OVF 1	OVF 0
W																
Reset	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c

Figure 8-8. Overrun Status Register (SIU_OSR)

Table 8-10. SIU_OSR Field Descriptions

Field	Function
OVF*n*	Overrun Flag *n.* This bit is set when an overrun occurs on the corresponding IRQ*n* pin. 0 No overrun occurred on the corresponding IRQ*n* pin. 1 An overrun occurred on the corresponding IRQ*n* pin.

8.3.2.8 Overrun Request Enable Register (SIU_ORER)

The SIU_ORER contains bits to enable an overrun if the corresponding flag bit is set in the SIU_OSR. If any overrun request enable bit and the corresponding flag bit are set, the single combined overrun request from the SIU to the interrupt controller is asserted.

Offset: SIU_BASE + 0x0024 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	ORE 15	ORE 14	ORE 13	ORE 12	ORE 11	ORE 10	ORE 9	ORE 8	ORE 7	ORE 6	ORE 5	ORE 4	ORE 3	ORE 2	ORE 1	ORE 0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 8-9. Overrun Request Enable Register (SIU_ORER)

Table 8-11. SIU_ORER Field Descriptions

Field	Function
ORE*n*	Overrun Request Enable *n.* Enables the corresponding overrun request when an overrun occurs on the corresponding IRQ*n* pin. 0 Overrun request disabled. 1 Overrun request enabled.

8.3.2.9 IRQ Rising-Edge Event Enable Register (SIU_IREER)

The SIU_IREER allows rising-edge-triggered events to be enabled on the corresponding IRQ*n* pins. Setting the corresponding bits in the SIU_IREER and SIU_IFEER enables rising- and falling-edge events.

Offset: SIU_BASE + 0x0028 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	NREE0[1]	NREE1[1]	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	IREE 15	IREE 14	IREE 13	IREE 12	IREE 11	IREE 10	IREE 9	IREE 8	IREE 7	IREE 6	IREE 5	IREE 4	IREE 3	IREE 2	IREE 1	IREE 0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

[1] Once written, the NREE*n* bits cannot be changed until the next reset.

Figure 8-10. IRQ Rising-Edge Event Enable Register (SIU_IREER)

Table 8-12. SIU_IREER Field Descriptions

Field	Function
NREE*n*	NMI Rising-Edge Event Enable *n.* These write-once bits enable rising-edge-triggered events on the corresponding NMI*n* input. 0 Rising edge event disabled. 1 Rising edge event enabled. **Note:** Once written, the NREE*n* bits cannot be changed until the next reset.
IREE*n*	IRQ Rising-Edge Event Enable *n.* Enables rising-edge triggered events on the corresponding IRQ*n* pin. 0 Rising edge event disabled. 1 Rising edge event enabled.

8.3.2.10 IRQ Falling-Edge Event Enable Register (SIU_IFEER)

The SIU_IFEER allows falling-edge-triggered events to be enabled on the corresponding IRQ*n* pins. Setting the corresponding bits in the SIU_IREER and SIU_IFEER enables rising- and falling-edge events.

Offset: SIU_BASE + 0x002C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	NFEE0[1]	NFEE1[1]	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	IFEE 15	IFEE 14	IFEE 13	IFEE 12	IFEE 11	IFEE 10	IFEE 9	IFEE 8	IFEE 7	IFEE 6	IFEE 5	IFEE 4	IFEE 3	IFEE 2	IFEE 1	IFEE 0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

[1] Once written, the NFEE*n* bits cannot be changed until the next reset.

Figure 8-11. IRQ Falling-Edge Event Enable Register (SIU_IFEER)

Table 8-13. SIU_IFEER Field Descriptions

Field	Function
NFEE*n*	NMI Falling-Edge Event Enable *n.* These write-once bits enable rising-edge triggered events on the corresponding NMI*n* input. 0 Falling edge event disabled. 1 Falling edge event enabled. **Note:** Once written, the NFEE*n* bits cannot be changed until the next reset.
IFEE*n*	IRQ Falling-Edge Event Enable *n.* Enables falling-edge triggered events on the corresponding IRQ*n* pin. 0 Falling edge event disabled. 1 Falling edge event enabled.

8.3.2.11 External IRQ Digital Filter Register (SIU_IDFR)

The SIU_IDFR specifies the amount of digital filtering on the IRQ0–IRQ15 pins. The digital filter length field specifies the number of system clocks that define the period of the digital filter and the minimum time a signal must be held in the active state on the IRQ pins to be recognized as an edge-triggered event.

Offset: SIU_BASE + 0x0030 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	DFL			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 8-12. External IRQ Digital Filter Register (SIU_IDFR)

Table 8-14. SIU_IDFR Field Descriptions

Field	Function
DFL	Digital Filter Length. Defines digital filter period on the IRQ*n* inputs according to the following equation: $\text{Filter Period} = (\text{SystemClockPeriod} \times 2^{\text{DFL}}) + 1(\text{SystemClockPeriod})$ For a 116 MHz system clock, this gives a range of 15.6 ns to 256 μs. The minimum time of two clocks accounts for synchronization of the IRQ input pins with the system clock.

8.3.2.12 IRQ Filtered Input Register (SIU_IFIR)

This is a read-only register that captures the output of the NMI*n* and IRQ*n* digital input filters.

Offset: SIU_BASE + 0x0034 Access: User read only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	FNMI0	FNMI1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	FI15	FI14	FI13	FI12	IFI11	FI10	FI9	FI8	FI7	FI6	FI5	FI4	FI3	FI2	FI1	FI0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 8-13. External IRQ Filtered Input Register (SIU_IFIR)

Table 8-15. SIU_IFIR Field Descriptions

Field	Function
FNMI0	Filtered Non-Maskable Interrupt 0. This bit is set/cleared for the corresponding NMI pin: 0 A logic one has passed through the NMI digital filter for NMI0 pin. 1 A logic zero has passed through the NMI digital filter for NMI0 pin.
FNMI1	Filtered Non-Maskable Interrupt 1. This bit is set/cleared for the corresponding NMI pin: 0 A logic one has passed through the NMI digital filter for NMI1 pin. 1 A logic zero has passed through the NMI digital filter for NMI1 pin.
FI*n*	Filtered Input *n*. This bit is set/cleared for the corresponding filtered IRQ pin: 0 A logic one has passed through the IRQ digital filter for the corresponding IRQ pin. 1 A logic zero has passed through the IRQ digital filter for the corresponding IRQ pin.

8.3.2.13 Pad Configuration Registers (SIU_PCR)

The following subsections define the SIU_PCRs for all device pins that allow configuration of the pin function, direction, and static electrical attributes. The information presented pertains to which bits and fields are active for a given pin or group of pins, and the register reset state. The reset state of SIU_PCRs in the following sections is prior to executing the boot-assist module (BAM) program. The BAM program may change SIU_PCRs based on reset configuration. See the BAM section of the manual for more detail.

For all SIU_PCRs:

- If the pin is configured as an input, the ODE and SRC bits do not apply.
- If the pin is configured as an output, the HYS bit does not apply.
- When a pin is configured as an output, the weak internal pull up/down is disabled, regardless of the WPE or WPS settings in the SIU_PCR.

IBE and OBE bit definitions are specific to each SIU_PCR. When an I/O function is input- or output-only, the IBE and OBE bits do not have to be set to enable the input or output. When an I/O function can be either an input and output, the IBE and OBE bits must be set accordingly (IBE = 1 for input, and OBE = 1 for output). For I/O functions that change direction dynamically, such as the MLBSIG and MLBDAT, switching between input and output is handled internally, and the IBE and OBE bits have no effect.

For all SIU_PCRs where GPIO function is available on the pin, if the pin is configured as an output and the IBE bit is set, the actual pin value is reflected in the corresponding GPDI*n* register. Negating the IBE bit when the pin is configured as an output reduces noise and power consumption. Reads from the GPDI*n* registers are undefined when the corresponding IBE bit is negated.

The SIU_PCRs are 16-bit registers that may be read or written as 32-bit values aligned on 32-bit address boundaries. Table 8-16 describes the SIU_PCR fields.

NOTE

Not all of the fields may be present in a given SIU_PCR, depending on the type of pad it controls. See the specific SIU_PCR definition.

For all SIU_PCRs, the associated pin supports GPIO and as many as three alternate functions. The PA field is defined in Table 8-16. For all SIU_PCRs of this type, a value of 0b11 selects Function 3, a value of 0b10 selects Function 2, the value 0b01 selects Function 1, and a value 0b00 selects GPIO.

All pins are named according to their associated parallel port name and associated bit number. For example, the Port A pins are named PA0 to PA15 (these pin names should not be confused with the PA bitfield, which is present in every SIU_PCR.) See Table 3-1 in Section 3.4, Detailed Signal Description, for a list of pins and their functions, including values for the PA bitfield for setting the function of each GPIO pin.

NOTE

Table 3-1 lists the available functions for each pin. Do not select reserved values for the PA bitfield.

Some SIU_PCRs contain a slew rate control (SRC) field. Slew rate control pertains to pins with slow or medium I/O pad types. The SRC field for all SIU_PCRs with slew rate control is defined in Table 8-16.

Table 8-16. SIU_PCR Field Descriptions

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>PA</td><td>Pin Assignment. Selects a multiplexed pad function. A separate port enable output signal from the SIU is asserted for each register value.
<table>
<tr><th>PA Field</th><th>Pin Function</th></tr>
<tr><td>0b00</td><td>GPIO</td></tr>
<tr><td>0b01</td><td>Function 1</td></tr>
<tr><td>0b10</td><td>Function 2</td></tr>
<tr><td>0b11</td><td>Function 3</td></tr>
</table></td></tr>
<tr><td>OBE</td><td>Output Buffer Enable. Enables the pad as an output and drives the output buffer enable signal.
0 Output buffer for the pad disabled.
1 Output buffer for the pad enabled.</td></tr>
<tr><td>IBE</td><td>Input Buffer Enable. Enables the pad as an input and drives the input buffer enable signal.
0 Input buffer for the pad disabled.
1 Input buffer for the pad enabled.</td></tr>
<tr><td>DSC</td><td>Drive Strength Control. Controls the drive strength control output signals from the SIU. The output signals are driven to the value of this field. The actual drive strengths are defined by the implementation of the pad devices for a given device.
<table>
<tr><th>DSC</th><th>Drive Strength</th></tr>
<tr><td>0b00</td><td>10 pF Drive Strength</td></tr>
<tr><td>0b01</td><td>20 pF Drive Strength</td></tr>
<tr><td>0b10</td><td>30 pF Drive Strength</td></tr>
<tr><td>0b11</td><td>50 pF Drive Strength</td></tr>
</table>
Note: DSC is applicable to fast pads only. See Table 3-1 in Section 3.4, Detailed Signal Description, for a listing of pad types.</td></tr>
<tr><td>ODE</td><td>Open Drain Output Enable. Controls output driver configuration for the pads. Either open drain or push/pull driver configurations can be selected. This feature applies only when pins are configured as outputs.
0 Open drain disabled for the pad (push/pull driver enabled).
1 Open drain enabled for the pad.</td></tr>
<tr><td>HYS</td><td>Input Hysteresis. Controls whether hysteresis is enabled for the pad.
0 Hysteresis disabled for the pad.
1 Hysteresis enabled for the pad.</td></tr>
</table>

Table 8-16. SIU_PCR Field Descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>SRC</td><td>Slew Rate Control. Controls slew rate for the pad. Slew rate control pertains to pins with slow or medium I/O pad types, and the output signals are driven according to the value of this field. Actual slew rate is dependent on the pad type and load. See the PXN20 Microcontroller Data Sheet for this information.
<table>
<tr><th>SRC</th><th>Slew Rate</th></tr>
<tr><td>0b00</td><td>Minimum slew rate (slowest)</td></tr>
<tr><td>0b01</td><td>Medium slew rate</td></tr>
<tr><td>0b10</td><td>Reserved</td></tr>
<tr><td>0b11</td><td>Maximum slew rate (fastest)</td></tr>
</table>
Note: SRC is applicable to slow or medium pads only. See Table 3-1 in Section 3.4, Detailed Signal Description, for a listing of pad types.</td></tr>
<tr><td>WPE</td><td>Weak Pullup/Down Enable. Controls whether the weak pullup/down devices are enabled/disabled for the pad.
0 Weak pull device is disabled for the pad.
1 Weak pull device is enabled for the pad.</td></tr>
<tr><td>WPS</td><td>Weak Pullup/Down Select. Controls whether weak pullup or weak pulldown devices are used for the pad when weak pullup/down devices are enabled.
0 Pulldown value enabled for the pad.
1 Pullup value enabled for the pad.</td></tr>
</table>

8.3.2.13.1 Pad Configuration Registers 0–15 (SIU_PCR0–SIU_PCR15)

The SIU_PCR0 to SIU_PCR15 registers control the pin function and static electrical attributes of the Port A pins PA0 to PA15 (input only). For each pin, Table 3-1 lists the signals available as the PA settings for Function1, Function2, and Function3.

Offset: SIU_BASE + 0x0040–SIU_BASE + 0x005E Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA		0	IBE	0	0	0	HYS	0	0	WPE	WPS
W																
Reset	0	0	0	0	0	0	0	0	0	0	0[1]	0	0	0	0	0

[1] A write to this bit has no effect. A read will return the written value.

Figure 8-14. Port A Pad Configuration Registers (SIU_PCR0–SIU_PCR15)

See Table 8-16 for bit field definitions.

8.3.2.13.2 Pad Configuration Registers 16–143 (SIU_PCR16–SIU_PCR143) and 147–154 (SIU_PCR147–SIU_PCR154)

The SIU_PCR16 to SIU_PCR143 and SIU_PCR147 to SIU_PCR154 registers control the pin function, direction, and static electrical attributes of the Port B (PB0–PB15), Port C (PC0–PC15), Port D (PD0–PD15), Port E (PE0–PE15), Port F (PF0–PF15), Port G (PG0–PG15), Port H (PH0–PH15), Port J (PJ0–PJ15), and Port K (PK0–PK10) pins, except for Port K pins PK[0:2], which are shown in

Section 8.3.2.13.3, Pad Configuration Registers 144–146 (SIU_PCR144–SIU_PCR146). For each pin, Table 3-1 lists the signals that are available as the PA settings for Function1, Function2, and Function3.

Offset: SIU_BASE+0x0060–SIU_BASE+0x015E; SIU_BASE+0x0166–SIU_BASE+0x0174 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA		OBE	IBE	0	0	ODE	HYS	SRC		WPE	WPS
W																
Reset	0	0	0	0	0	0	0	U[1]	0	0	0	0	0	0	0[1]	0

1 The reset value is 1 for SIU_PCR153 (BOOTCFG), 0 for all other SIU_PCRs in this range.

Figure 8-15. Port B to Port K Pad Config Registers (SIU_PCR16–SIU_PCR143, SIU_PCR147–SIU_PCR154)

See Table 8-16 for bit field definitions.

8.3.2.13.3 Pad Configuration Registers 144–146 (SIU_PCR144–SIU_PCR146)

The SIU_PCR144 to SIU_PCR146 registers control the pin function, direction, and static electrical attributes of the Port K pins 0–2 (PK0–PK2). For each pin, Table 3-1 lists the signals that are available as the PA settings for Function1, Function2, and Function3.

Offset: SIU_BASE+0x0160–SIU_BASE+0x0164 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA		OBE	IBE	DSC[1]		ODE	HYS	0	0	WPE	WPS
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

1 When using PK[0:2] for MLB (PA = 0b01), the recommended value for DSC is 0b11.

Figure 8-16. Port K[0:2] Pad Configuration Registers (SIU_PCR144–SIU_PCR146)

See Table 8-16 for bit field definitions.

8.3.2.14 GPIO Pin Data Output Registers (SIU_GPDO16_19–SIU_GPDO152_154)

The SIU_GPDO16_19 register definition is in Figure 8-17. All other SIU_GPDO*n* registers follow the same pattern where four GPDO bits are placed in a 32-bit word, with one bit per byte. Each of the 139 PDO bits corresponds to a port pin in the order given in Table 8-18. Gaps exist in this memory space where the pin is not available in the package.

NOTE

On PXN20, the Port A pins are only general-purpose inputs. Therefore, there are no output data registers associated with these pins.

The SIU_GPDO*n* registers are written to by software to drive data out on the external GPIO pin. Each byte of a register drives a single external GPIO pin, which allows the pin state to be controlled independently from other GPIO pins. Writes to the SIU_GPDO*n* registers do not affect pin states if the pins are configured as inputs or as non-GPIO function by the associated pad configuration registers. The SIU_GPDO*n* register values are automatically driven to the GPIO pins without software update if the GPIO pins' direction changes from input to output.

Offset: SIU_BASE + 0x0610–SIU_BASE+0x0698 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	PDO 16	0	0	0	0	0	0	0	PDO 17
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	PDO 18	0	0	0	0	0	0	0	PDO 19
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 8-17. GPIO Pin Data Out Register 16–19 (SIU_GPDO16_19)

Table 8-17. SIU_GPDO*n* Field Descriptions

Field	Description
PDO*n*	Pin Data Out. Stores the data to be driven out on the external GPIO pin associated with the register. If the register is read, it returns the value written. 0 V_{OL} driven on the external GPIO pin when the pin is configured as an output. 1 V_{OH} driven on the external GPIO pin when the pin is configured as an output.

Table 8-18. Pin Data Output Register to Pin Mapping

SIU_GPDO*n*	Address Offset	Pins
16_19 20_23 24_27 28_31	0x0610 0x0614 0x0618 0x061C	PB0–PB3 PB4–PB7 PB8–PB11 PB12–PB15
32_35 36_39 40_43 44_47	0x0620 0x0624 0x0628 0x062C	PC0–PC3 PC4–PC7 PC8–PC11 PC12–PC15
48_51 52_55 56_59 60_63	0x0630 0x0634 0x0638 0x063C	PD0–PD3 PD4–PD7 PD8–PD11 PD12–PD15
64_67 68_71 72_75 76_79	0x0640 0x0644 0x0648 0x064C	PE0–PE3 PE4–PE7 PE8–PE11 PE12–PE15
80_83 84_87 88_91 92_95	0x0650 0x0654 0x0658 0x065C	PF0–PF3 PF4–PF7 PF8–PF11 PF12–PF15
96_99 100_103 104_107 108_111	0x0660 0x0664 0x0668 0x066C	PG0–PG3 PG4–PG7 PG8–PG11 PG12–PG15

Table 8-18. Pin Data Output Register to Pin Mapping *(continued)*

SIU_GPDO*n*	Address Offset	Pins
112_115 116_119 120_123 124_127	0x0670 0x0674 0x0678 0x067C	PH0–PH3 PH4–PH7 PH8–PH11 PH12–PH15
128_131 132_135 136_139 140_143	0x0680 0x0684 0x0688 0x068C	PJ0–PJ3 PJ4–PJ7 PJ8–PJ11 PJ12–PJ15
144_147 148_151 152_154	0x0690 0x0694 0x0698	PK0–PK3 PK4–PK7 PK8–PK10

8.3.2.15 GPIO Pin Data Input Registers (SIU_GPDI0_3–SIU_GPDI152_154)

The definition of the SIU_GPDI0_3 register is given in Figure 8-18. All other SIU_GPDI*n* registers follow the same pattern where 4 GPDI bits are placed in a 32-bit word, with one bit per byte. Each of the 155 GPDI bits correspond to the port pin (Table 8-20). Gaps exist in this memory space where the pin is not available in the package.

The SIU_GPDI*n* registers are read-only registers that allow software to read the input state of an external GPIO pin. Each byte of a register represents the input state of a single external GPIO pin. If the GPIO pin is configured as an output, and the input buffer enable (IBE) bit is set in the associated Pad Configuration Register, the SIU_GPDI*n* register reflects the actual state of the output pin.

Offset: SIU_BASE + 0x0800–SIU_BASE+0x0891 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	PDI0	0	0	0	0	0	0	0	PDI1
W																
Reset	0	0	0	0	0	0	0	U	0	0	0	0	0	0	0	U

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	PDI2	0	0	0	0	0	0	0	PDI3
W																
Reset	0	0	0	0	0	0	0	U	0	0	0	0	0	0	0	U

Figure 8-18. GPIO Pin Data Input Register 0–3 (SIU_GPDI0_3)

Table 8-19. SIU_GPDI*n* Field Description

Field	Description
PDI*n*	Pin Data In. This bit reflects the input state on the external GPIO pin associated with the register. 0 Signal on pin is less than or equal to V_{IL}. 1 Signal on pin is greater than or equal to V_{IH}.

Table 8-20. GPIO Pin Data Input Register to Pin Mapping

SIU_GPDI*n*	Address Offset	Pins
0_3	0x0800	PA0–PA3
4_7	0x0804	PA4–PA7
8_11	0x0808	PA8–PA11
12_15	0x080C	PA12–PA15
16_19	0x0810	PB0–PB3
20_23	0x0814	PB4–PB7
24_27	0x0818	PB8–PB11
28_31	0x081C	PB12–PB15
32_35	0x0820	PC0–PC3
36_39	0x0824	PC4–PC7
40_43	0x0828	PC8–PC11
44_47	0x082C	PC12–PC15
48_51	0x0830	PD0–PD3
52_55	0x0834	PD4–PD7
56_59	0x0838	PD8–PD11
60_63	0x083C	PD12–PD15
64_67	0x0840	PE0–PE3
68_71	0x0844	PE4–PE7
72_75	0x0848	PE8–PE11
76_79	0x084C	PE12–PE15
80_83	0x0850	PF0–PF3
84_87	0x0854	PF4–PF7
88_91	0x0858	PF8–PF11
92_95	0x085C	PF12–PF15
96_99	0x0860	PG0–PG3
100_103	0x0864	PG4–PG7
104_107	0x0868	PG8–PG11
108_111	0x086C	PG12–PG15
112_115	0x0870	PH0–PH3
116_119	0x0874	PH4–PH7
120_123	0x0878	PH8–PH11
124_127	0x087C	PH12–PH15
128_131	0x0880	PJ0–PJ3
132_135	0x0884	PJ4–PJ7
136_139	0x0888	PJ8–PJ11
140_143	0x088C	PJ12–PJ15
144_147	0x0890	PK0–PK3
148_151	0x0894	PK4–PK7
152_154	0x0898	PK8–PK10

8.3.2.16 IMUX Select Register 1 (SIU_ISEL1)

The SIU_ISEL1 selects the source for the external interrupt. The selection is made in conjunction with SIU_ISEL2 and SIU_ISEL2A registers. Figure 8-72 shows how ISEL1, ISEL2, and ISEL2A interact.

Offset: SIU_BASE + 0x0904 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	ESEL15		ESEL14		ESEL13		ESEL12		ESEL11		ESEL10		ESEL9		ESEL8	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	ESEL7		ESEL6		ESEL5		ESEL4		ESEL3		ESEL2		ESEL1		ESEL0	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 8-19. IMUX Select Register 1 (SIU_ISEL1)

Table 8-21. SIU_ISEL1 Field Descriptions

Field	Description[1]
ESEL15	External IRQ Input Select 15. Specifies input for IRQ15. 00 PB15 pin. 01 PC15 pin. 10 PD15 pin. 11 ISEL2.
ESEL14	External IRQ Input Select 14. Specifies input for IRQ14. 00 PB14 pin. 01 PC14 pin. 10 PD14 pin. 11 ISEL2.
ESEL13	External IRQ Input Select 13. Specifies input for IRQ13. 00 PB13 pin. 01 PC13 pin. 10 PD13 pin. 11 ISEL2.
ESEL12	External IRQ Input Select 12. Specifies input for IRQ12. 00 PB12 pin. 01 PC12 pin. 10 PD12 pin. 11 ISEL2.
ESEL11	External IRQ Input Select 11. Specifies input for IRQ11. 00 PB11 pin. 01 PC11 pin. 10 PD11 pin. 11 ISEL2.
ESEL10	External IRQ Input Select 10. Specifies input for IRQ10. 00 PB10 pin. 01 PC10 pin. 10 PD10 pin. 11 ISEL2.

Table 8-21. SIU_ISEL1 Field Descriptions (continued)

Field	Description[1]
ESEL9	External IRQ Input Select 9. Specifies input for IRQ9. 00 PB9 pin. 01 PC9 pin. 10 PD9 pin. 11 ISEL2.
ESEL8	External IRQ Input Select 8. Specifies input for IRQ8. 00 PB8 pin. 01 PC8 pin. 10 PD8 pin. 11 ISEL2.
ESEL7	External IRQ Input Select 7. Specifies input for IRQ7. 00 PB7 pin. 01 PC7 pin. 10 PD7 pin. 11 ISEL2.
ESEL6	External IRQ Input Select 6. Specifies input for IRQ6. 00 PB6 pin. 01 PC6 pin. 10 PD6 pin. 11 ISEL2.
ESEL5	External IRQ Input Select 5. Specifies input for IRQ5. 00 PB5 pin. 01 PC5 pin. 10 PD5 pin. 11 ISEL2.
ESEL4	External IRQ Input Select 4. Specifies input for IRQ4. 00 PB4 pin. 01 PC4 pin. 10 PD4 pin. 11 ISEL2.
ESEL3	External IRQ Input Select 3. Specifies input for IRQ3. 00 PB3 pin. 01 PC3 pin. 10 PD3 pin. 11 ISEL2.
ESEL2	External IRQ Input Select 2. Specifies input for IRQ2. 00 PB2 pin. 01 PC2 pin. 10 PD2 pin. 11 ISEL2.

Table 8-21. SIU_ISEL1 Field Descriptions (continued)

Field	Description[1]
ESEL1	External IRQ Input Select 1. Specifies input for IRQ1. 00 PB1 pin. 01 PC1 pin. 10 PD1 pin. 11 ISEL2.
ESEL0	External IRQ Input Select 0. Specifies input for IRQ0. 00 PB0 pin. 01 PC0 pin. 10 PD0 pin. 11 ISEL2.

1 Pins specified in this table must be configured as general purpose inputs to be used as external IRQs.

8.3.2.17 IMUX Select Register 2 (SIU_ISEL2)

The SIU_ISEL2 register selects the source for the external interrupt. The selection is made in conjunction with SIU_ISEL1 and SIU_ISEL2A. Figure 8-72 shows how ISEL1, ISEL2, and ISEL2A interact.

Offset: SIU_BASE + 0x0908 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	ESEL15		ESEL14		ESEL13		ESEL12		ESEL11		ESEL10		ESEL9		ESEL8	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	ESEL7		ESEL6		ESEL5		ESEL4		ESEL3		ESEL2		ESEL1		ESEL0	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 8-20. IMUX Select Register 2 (SIU_ISEL2)

Table 8-22. SIU_ISEL2 Field Descriptions

Field	Description
ESEL15	External IRQ Input Select 15. Specifies input for IRQ15. 00 PE15 pin. 01 PF15 pin. 10 PG15 pin. 11 ISEL2A.
ESEL14	External IRQ Input Select 14. Specifies input for IRQ14. 00 PE14 pin. 01 PF14 pin. 10 PG14 pin. 11 ISEL2A.
ESEL13	External IRQ Input Select 13. Specifies input for IRQ13. 00 PE13 pin. 01 PF13 pin. 10 PG13 pin. 11 ISEL2A.
ESEL12	External IRQ Input Select 12. Specifies input for IRQ12. 00 PE12 pin. 01 PF12 pin. 10 PG12 pin. 11 ISEL2A.
ESEL11	External IRQ Input Select 11. Specifies input for IRQ11. 00 PE11 pin. 01 PF11 pin. 10 PG11 pin. 11 ISEL2A.
ESEL10	External IRQ Input Select 10. Specifies input for IRQ10. 00 PE10 pin. 01 PF10 pin. 10 PG10 pin. 11 ISEL2A.

Table 8-22. SIU_ISEL2 Field Descriptions (continued)

Field	Description
ESEL9	External IRQ Input Select 9. Specifies input for IRQ9. 00 PE9 pin. 01 PF9 pin. 10 PG9 pin. 11 ISEL2A.
ESEL8	External IRQ Input Select 8. Specifies input for IRQ8. 00 PE8 pin. 01 PF8 pin. 10 PG8 pin. 11 ISEL2A.
ESEL7	External IRQ Input Select 7. Specifies input for IRQ7. 00 PE7 pin. 01 PF7 pin. 10 PG7 pin. 11 ISEL2A.
ESEL6	External IRQ Input Select 6. Specifies input for IRQ6. 00 PE6 pin. 01 PF6 pin. 10 PG6 pin. 11 ISEL2A.
ESEL5	External IRQ Input Select 5. Specifies input for IRQ5. 00 PE5 pin. 01 PF5 pin. 10 PG5 pin. 11 ISEL2A.
ESEL4	External IRQ Input Select 4. Specifies input for IRQ4. 00 PE4 pin. 01 PF4 pin. 10 PG4 pin. 11 ISEL2A.
ESEL3	External IRQ Input Select 3. Specifies input for IRQ3. 00 PE3 pin. 01 PF3 pin. 10 PG3 pin. 11 ISEL2A.
ESEL2	External IRQ Input Select 2. Specifies input for IRQ2. 00 PE2 pin. 01 PF2 pin. 10 PG2 pin. 11 ISEL2A.

Table 8-22. SIU_ISEL2 Field Descriptions (continued)

Field	Description
ESEL1	External IRQ Input Select 1. Specifies input for IRQ1. 00 PE1 pin. 01 PF1 pin. 10 PG1 pin. 11 ISEL2A.
ESEL0	External IRQ Input Select 0. Specifies input for IRQ0. 00 PE0 pin. 01 PF0 pin. 10 PG0 pin. 11 ISEL2A.

8.3.2.18 IMUX Select Register 4 (SIU_ISEL4)

The SIU_ISEL4 register specifies the source for the trigger input for ADC.

Offset: SIU_BASE + 0x0910 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	TSEL1							0	TSEL0						
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 8-21. ADC Trigger Input Select Register 4 (SIU_ISEL4)

Table 8-23. SIU_ISEL4 Field Descriptions

Field	Description
TSEL0	ADC External Trigger for start of conversion. 000_0000 PE10. 000_0001 PE11. 000_0010 PE12. 000_0011 PE13. 000_0100 PIT2. 000_0101 – 111_1111 Reserved (default is PE10).
TSEL1	Trigger for injected trigger. 000_0000 PE10. 000_0001 PE11. 000_0010 PE12. 000_0011 PE13. 000_0100 PIT2. 000_0101 – 111_1111 Reserved (default is PE10).

8.3.2.19 Chip Configuration Register (SIU_CCR)

Offset: SIU_BASE + 0x0980 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	MATCH	DISNEX
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	U	U

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	TEST LOCK	0	0	0	0	0	0[1]	0[2]
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

1 Writes to this bit have no effect, but reads return the written value.

2 Reserved, do not write.

Figure 8-22. Chip Configuration Register (SIU_CCR)

Table 8-24. SIU_CCR Field Descriptions

Field	Description
MATCH	Compare Register Match. The MATCH bit is a read-only bit that holds the value of the match input signal to the SIU. The match input is asserted if the serial boot password provided by the user matches the password stored in the flash. 0 Match input signal is negated. 1 Match input signal is asserted.
DISNEX	Disable Nexus. The DISNEX bit is a read-only bit that holds the value of the Nexus disable input signal to the SIU. When system reset negates, the value in this bit depends on the censorship control word and the boot configuration bits. 0 Nexus disable input signal negated. 1 Nexus disable input signal asserted.
TESTLOCK	TEST Lock. The TESTLOCK bit prevents access to Freescale internal test features. These internal test features are enabled by writing to reserved test bits in the device. Setting the TESTLOCK bit locks the test bits so that they cannot be changed inadvertently by runaway code. Customer initialization code should always set this bit. 0 Internal test features could be enabled. 1 Internal test features are disabled.

8.3.2.20 External Clock Control Register (SIU_ECCR)

The SIU_ECCR controls the timing relationship between the system clock and the external clocks, CLKOUT. All bits and fields in the SIU_ECCR are read/write and reset by the asynchronous reset signal.

Offset: SIU_BASE + 0x0984 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0[1]	0[1]	0[1]	0[1]	0[1]	0[1]	0[1]	0[1]	0[1]	0	0	0	ECEN	0	ECDF	
W																
Reset	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1

1 Writes to this bit have no effect, but reads return the written value.

Figure 8-23. External Clock Control Register (SIU_ECCR)

Table 8-25. SIU_ECCR Field Descriptions

Field	Description
ECEN	External Clock Enable. The ECEN bit enables CLKOUT. The CLKOUT waveform is determined by ECDF divides relative to the internal system clock. **Note:** To correctly reflect the CLKOUT waveform to the external pin, the SIU_PCR for the CLKOUT pin needs to be configured. 0 Disable CLKOUT waveform. 1 Enable CLKOUT waveform.
ECDF	External Clock Division Factor. Specifies frequency ratio between system clock and external clock, CLKOUT. The CLKOUT frequency is divided from the system clock frequency according to the descriptions below. 00 Divide by 1. 01 Divide by 2 (default value). 10 Divide by 4. 11 Divide by 8. **Note:** If ECDF is equal to 0b00 and SIU_SYSCLK[SYSCLKDIV] is not equal to 0b000, then the CLKOUT pin will not have a nominal 50% duty cycle.

8.3.2.21 General Purpose Register 0–3 (SIU_GPR*n*)

The SIU_GPR*n* registers provide general-purpose read/write registers for customer use.

Offset: SIU_BASE + 0x0988 (SIU_GPR0), 0x098C (SIU_GPR1), 0x0990 (SIU_GPR2), 0x0994 (SIU_GPR3) — Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	GP															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	GP															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 8-24. General Purpose Register 0–3 (SIU_GPR*n*)

8.3.2.22 System Clock Register (SIU_SYSCLK)

The SIU_SYSCLK register controls the source for the system clock, the divider for the system clock, and eight fields that control the clock divider for groups of peripherals. For a listing of which peripherals are associated with which LPCLKDIV bit on PXN20, see Section 5.3.5, Peripheral Clock Dividers.

Offset: SIU_BASE + 0x09A0 — Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	SYSCLKSEL		SYSCLKDIV			0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	LPCLKDIV3		LPCLKDIV2		LPCLKDIV1		LPCLKDIV0	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 8-25. System Clock Register (SIU_SYSCLK)

Table 8-26. SIU_SYSCLK Field Descriptions

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>SYSCLKSEL</td><td>System Clock Select. The SYSCLKSEL bit selects the source for the system clock.
00 System clock supplied by 16 MHz IRC.
01 System clock supplied by 4 – 40 MHz_XTAL.
10 System clock supplied by FMPLL.
11 Reserved (defaults to 16 MHz IRC).
Note: The default SYSCLKSEL value may be modified by the BAM code execution to 0b01 to select the 4–40 MHz XTAL as the system clock source to support the serial download operation. Please see Chapter 9, Boot Assist Module (BAM), for more details.</td></tr>
<tr><td>SYSCLKDIV</td><td>System Clock Divide. The SYSCLKDIV bits select the divider value for the system clock. The SYSCLKDIV divider is required in addition to the RFD to allow the other sources for the system clock (16 MHz IRC and 4 – 40 MHz XTAL) to be divided to slowest frequencies to improve power consumption.
000 Divide by 1.
001 Divide by 2.
010 Divide by 4.
011 Divide by 8.
100 Divide by 16.
101 – 111 Reserved (defaults to divide by 1).</td></tr>
<tr><td>LPCLKDIVn</td><td>Low-Power Peripheral Clock Divides. The LPCLKDIV bits select the divider values for each peripheral group.
00 Divide by 1.
01 Divide by 2.
10 Divide by 4.
11 Divide by 8.
<table>
<tr><th>LPCLKDIVn</th><th>Peripheral Set Number</th><th>Peripheral Groups</th></tr>
<tr><td>LPCLKDIV0</td><td>Peripheral Set 1</td><td>eSCI, I^2C</td></tr>
<tr><td>LPCLKDIV1</td><td>Peripheral Set 2</td><td>FlexCAN, SPI</td></tr>
<tr><td>LPCLKDIV2</td><td>Peripheral Set 3</td><td>ADC, CTU</td></tr>
<tr><td>LPCLKDIV3</td><td>Peripheral Set 4</td><td>eMIOS</td></tr>
</table></td></tr>
</table>

8.3.2.23 Halt Register (SIU_HLT*n*)

The SIU_HLT*n* register is used to halt the various peripherals by disabling the clocks to the module. Writing a 1 to a HLT*n* bit drives a separate halt request to the associated peripheral. After completing any task in progress, the peripheral shuts down its clock input and signals to the SIU_HLTACK*n* register that it has halted. Writing a 0 to a HLT*n* bit drives a separate request to the associated peripheral causes the peripheral to restart its clock input and signals to the SIU_HLTACK*n* register that it has restarted.

Writes to bits in SIU_HLT*n* that are not associated with a peripheral are reflected in the SIU_HLT*n* register and in the SIU_HLTACK*n* register, but have no other effect.

Offset: SIU_BASE + 0x09A4 Access: User read-only

	0[1]	1[1]	2[1]	3[1]	4[1]	5[1]	6	7	8[1]	9	10	11	12	13	14	15
R	0	0	0	0	0	0	HLT 6	HLT 7	0	HLT 9	HLT 10	HLT 11	HLT 12	HLT 13	HLT 14	HLT 15
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24[1]	25[1]	26	27	28	29	30[1]	31
R	HLT 16	HLT 17	HLT 18	HLT 19	HLT 20	HLT 21	HLT 22	HLT 23	0	0	HLT 26	HLT 27	HLT 28	HLT 29	0	HLT 31
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

1 Writes to this bit are reflected in the SIU_HLT0 and SIU_HLTACK0 register, but have no other effect.

Figure 8-26. Halt Register 0 (SIU_HLT0)

Table 8-27. SIU_HLT0 Register Field Descriptions

Field	Description
HLT6	Halt bit 6. Setting this bit halts the EMIOS200 module.
HLT7	Halt bit 7. Setting this bit halts the PIT module.
HLT9	Halt bit 9. Setting this bit halts the CTU module.
HLT10	Halt bit 10. Setting this bit halts the FLEXCAN_F module.
HLT11	Halt bit 11. Setting this bit halts the FLEXCAN_E module.
HLT12	Halt bit 12. Setting this bit halts the FLEXCAN_D module.
HLT13	Halt bit 13. Setting this bit halts the FLEXCAN_C module.
HLT14	Halt bit 14. Setting this bit halts the FLEXCAN_B module.
HLT15	Halt bit 15. Setting this bit halts the FLEXCAN_A module.
HLT16	Halt bit 16. Setting this bit halts the ESCI_H module.
HLT17	Halt bit 17. Setting this bit halts the ESCI_G module.
HLT18	Halt bit 18. Setting this bit halts the ESCI_F module.
HLT19	Halt bit 19. Setting this bit halts the ESCI_E module.
HLT20	Halt bit 20. Setting this bit halts the ESCI_D module.
HLT21	Halt bit 21. Setting this bit halts the ESCI_C module.
HLT22	Halt bit 22. Setting this bit halts the ESCI_B module.
HLT23	Halt bit 23. Setting this bit halts the ESCI_A module.
HLT26	Halt bit 26. Setting this bit halts the DSPI_B module.
HLT27	Halt bit 27. Setting this bit halts the DSPI_A module.
HLT28	Halt bit 28. Setting this bit halts the I^2C_B module.
HLT29	Halt bit 29. Setting this bit halts the I^2C_A module.
HLT31	Halt bit 31. Setting this bit halts the ADC module.

Offset: SIU_BASE + 0x09A8 Access: User read-only

	0[1]	1[1]	2[2]	3	4	5[2]	6[2]	7[2]	8[2]	9[2]	10[2]	11[2]	12[2]	13[2]	14[2]	15[2]
R	0	0	0	HLT 3	HLT 4	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16[2]	17[2]	18[2]	19[2]	20	21	22	23	24[2]	25[2]	26	27	28	29	30[2]	31[2]
R	0	0	0	0	HLT 20	HLT 21	HLT 22	HLT 23	0	0	HLT 26	HLT 27	HLT 28	HLT 29	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

1 Reserved, do not write to this bit.

2 Writes to this bit are reflected in the SIU_HLT1 and SIU_HLTACK1 register, but have no other effect.

Figure 8-27. Halt Register 1 (SIU_HLT1)

Table 8-28. SIU_HLT1 Register Field Descriptions

Field	Description
HLT3	Halt bit 3. Setting this bit halts the DMA module.
HLT4	Halt bit 4. Setting this bit halts the NPC module.
HLT20	Halt bit 20. Setting this bit halts the ESCI_M module.
HLT21	Halt bit 21. Setting this bit halts the ESCI_L module.
HLT22	Halt bit 22. Setting this bit halts the ESCI_K module.
HLT23	Halt bit 23. Setting this bit halts the ESCI_J module.
HLT26	Halt bit 26. Setting this bit halts the DSPI_D module.
HLT27	Halt bit 27. Setting this bit halts the DSPI_C module.
HLT28	Halt bit 28. Setting this bit halts the I^2C_D module.
HLT29	Halt bit 29. Setting this bit halts the I^2C_C module.

8.3.2.24 Halt Acknowledge Register (SIU_HLTACK*n*)

The SIU_HLTACK*n* bits indicate that the peripheral requested to halt via the HLT*n* bit has completed its halt process and has entered a halted state with the peripheral clocks disabled. The HLTACK*n* bits are read-only; writes have no effect.

Offset: SIU_BASE + 0x09AC Access: User read-only

	0[1]	1[1]	2[1]	3[1]	4[1]	5[1]	6	7	8[1]	9	10	11	12	13	14	15
R	0	0	0	0	0	0	HLT ACK 6	HLT ACK 7	0	HLT ACK 9	HLT ACK 10	HLT ACK 11	HLT ACK 12	HLT ACK 13	HLT ACK 14	HLT ACK 15
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24[1]	25[1]	26	27	28	29	30[1]	31
R	HLT ACK 16	HLT CK 17	HLT ACK 18	HLT ACK 19	HLT ACK 20	HLT ACK 21	HLT ACK 22	HLT ACK 23	0	0	HLT ACK 26	HLT ACK 27	HLT ACK 28	HLT ACK 29	0	HLT ACK 31
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

1 Setting the corresponding bit in SIU_HLT0 sets this bit, but has no other effect.

Figure 8-28. Halt Acknowledge Register 0 (SIU_HLTACK0)

Table 8-29. SIU_HLTACK0 Register Field Descriptions

Field	Description
HLTACK6	Halt acknowledge bit 6. When this bit is set, the EMIOS200 module is halted.
HLTACK7	Halt acknowledge bit 7. When this bit is set, the PIT module is halted.
HLTACK9	Halt acknowledge bit 9. When this bit is set, the CTU module is halted.
HLTACK10	Halt acknowledge bit 10. When this bit is set, the FLEXCAN_F module is halted.
HLTACK11	Halt acknowledge bit 11. When this bit is set, the FLEXCAN_E module is halted.
HLTACK12	Halt acknowledge bit 12. When this bit is set, the FLEXCAN_D module is halted.
HLTACK13	Halt acknowledge bit 13. When this bit is set, the FLEXCAN_C module is halted.
HLTACK14	Halt acknowledge bit 14. When this bit is set, the FLEXCAN_B module is halted.
HLTACK15	Halt acknowledge bit 15. When this bit is set, the FLEXCAN_A module is halted.
HLTACK16	Halt acknowledge bit 16. When this bit is set, the ESCI_H module is halted.
HLTACK17	Halt acknowledge bit 17. When this bit is set, the ESCI_G module is halted.
HLTACK18	Halt acknowledge bit 18. When this bit is set, the ESCI_F module is halted.
HLTACK19	Halt acknowledge bit 19. When this bit is set, the ESCI_E module is halted.
HLTACK20	Halt acknowledge bit 20. When this bit is set, the ESCI_D module is halted.
HLTACK21	Halt acknowledge bit 21. When this bit is set, the ESCI_C module is halted.
HLTACK22	Halt acknowledge bit 22. When this bit is set, the ESCI_B module is halted.
HLTACK23	Halt acknowledge bit 23. When this bit is set, the ESCI_A module is halted.
HLTACK26	Halt acknowledge bit 26. When this bit is set, the DSPI_B module is halted.
HLTACK27	Halt acknowledge bit 27. When this bit is set, the DSPI_A module is halted.
HLTACK28	Halt acknowledge bit 28. When this bit is set, the I^2C_B module is halted.

Table 8-29. SIU_HLTACK0 Register Field Descriptions (continued)

Field	Description
HLTACK29	Halt acknowledge bit 29. When this bit is set, the I^2C_A module is halted.
HLTACK31	Halt acknowledge bit 31. When this bit is set, the ADC module is halted.

Offset: SIU_HLTACK1: SIU_BASE + 0x09B0 Access: User read-only

	0	1	2[1]	3	4	5[1]	6[1]	7[1]	8[1]	9[1]	10[1]	11[1]	12[1]	13[1]	14[1]	15[1]
R	HLTACK0	HLTACK1	0	HLTACK3	HLTACK4	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16[1]	17[1]	18[1]	19[1]	20	21	22	23	24[1]	25[1]	26	27	28	29	30[1]	31[1]
R	0	0	0	0	HLTACK20	HLTACK21	HLTACK22	HLTACK23	0	0	HLTACK26	HLTACK27	HLTACK28	HLTACK29	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

[1] Setting the corresponding bit in SIU_HLT0 sets this bit, but has no other effect.

Figure 8-29. Halt Acknowledge Register 1 (SIU_HLTACK1)

Table 8-30. SIU_HLTACK1 Register Field Descriptions

Field	Description
HLTACK0	Halt acknowledge bit 0. When this bit is set, the Z6 core is halted. **Note:** This flag indicates a core-generated halt, not a halt caused by writing to SIU_HLT10[HLT0].
HLTACK1	Halt acknowledge bit 1. When this bit is set, the Z0 core is halted. **Note:** This flag indicates a core-generated halt, not a halt caused by writing to SIU_HLT1[HLT1].
HLTACK3	Halt acknowledge bit 3. When this bit is set, the DMA module is halted.
HLTACK4	Halt acknowledge bit 4. When this bit is set, the NPC module is halted.
HLTACK20	Halt acknowledge bit 20. When this bit is set, the ESCI_M module is halted.
HLTACK21	Halt acknowledge bit 21. When this bit is set, the ESCI_L module is halted.
HLTACK22	Halt acknowledge bit 22. When this bit is set, the ESCI_K module is halted.
HLTACK23	Halt acknowledge bit 23. When this bit is set, the ESCI_J module is halted.
HLTACK26	Halt acknowledge bit 26. When this bit is set, the DSPI_D module is halted.
HLTACK27	Halt acknowledge bit 27. When this bit is set, the DSPI_C module is halted.
HLTACK28	Halt acknowledge bit 28. When this bit is set, the I^2C_D module is halted.
HLTACK29	Halt acknowledge bit 29. When this bit is set, the I^2C_C module is halted.

8.3.2.25 eMIOS Select Register *n* (SIU_EMIOS_SEL*n*)

The SIU_EMIOS_SEL*n* register specifies the source for the eMIOS[31:0] input channels, thus allowing the timer input channels to come from the pins, or from the deserialized output of one of the four DSPI modules. Each 4-bit field (32 fields across the four SIU_EMIOS_SEL*n* registers) in this set of registers individually controls the setting for one eMIOS input channel.

Offset: SIU_BASE + 0x09B4 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	EMIOSSEL31				EMIOSSEL30				EMIOSSEL29				EMIOSSEL28			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	EMIOSSEL27				EMIOSSEL26				EMIOSSEL25				EMIOSSEL24			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 8-30. eMIOS Select Register 0 (SIU_EMIOS_SEL0)

Offset: SIU_BASE + 0x09B8 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	EMIOSSEL23				EMIOSSEL22				EMIOSSEL21				EMIOSSEL20			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	EMIOSSEL19				EMIOSSEL18				EMIOSSEL17				EMIOSSEL16			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 8-31. eMIOS Select Register 1 (SIU_EMIOS_SEL1)

Offset: SIU_BASE + 0x09BC Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	EMIOSSEL15				EMIOSSEL14				EMIOSSEL13				EMIOSSEL12			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	EMIOSSEL11				EMIOSSEL10				EMIOSSEL9				EMIOSSEL8			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 8-32. eMIOS Select Register 2 (SIU_EMIOS_SEL2)

Offset: SIU_BASE + 0x09C0 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	EMIOSSEL7				EMIOSSEL6				EMIOSSEL5				EMIOSSEL4			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	EMIOSSEL3				EMIOSSEL2				EMIOSSEL1				EMIOSSEL0			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 8-33. eMIOS Select Register 3 (SIU_EMIOS_SEL3)

Table 8-31. SIU_EMIOS_SEL*n* Field Descriptions

Field	Description
EMIOSSEL*n*	eMIOS Channel[*n*] connection options. 0000–0011 eMIOS[*n*] input pin. 0100 DSPI_A deserialized output. 0101 DSPI_B deserialized output. 0110 DSPI_C deserialized output. 0111 DSPI_D deserialized output. 1000–1111 eMIOS[*n*] input pin.

8.3.2.26 External Interrupt Select Register 2A (SIU_ISEL2A)

The SIU_ISEL2A register selects the source for the external interrupts. Figure 8-72 shows how ISEL1, ISEL2, and ISEL2A interact.

Offset: SIU_BASE + 0x09C4 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	ESEL15		ESEL14		ESEL13		ESEL12		ESEL11		ESEL10		ESEL9		ESEL8	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	ESEL7		ESEL6		ESEL5		ESEL4		ESEL3		ESEL2		ESEL1		ESEL0	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 8-34. External Interrupt Select Register 2A (SIU_ISEL2A)

Table 8-32. SIU_ISEL2A Field Descriptions

Field	Description
ESEL15	External IRQ Input Select 15. Specifies input for IRQ15. 00 PH15 pin. 01 PJ15 pin. 10 Reserved. 11 Reserved.
ESEL14	External IRQ Input Select 14. Specifies input for IRQ14. 00 PH14 pin. 01 PJ14 pin. 10 Reserved. 11 Reserved.
ESEL13	External IRQ Input Select 13. Specifies input for IRQ13. 00 PH13 pin. 01 PJ13 pin. 10 Reserved. 11 Reserved.
ESEL12	External IRQ Input Select 12. Specifies input for IRQ12. 00 PH12 pin. 01 PJ12 pin. 10 Reserved. 11 Reserved.
ESEL11	External IRQ Input Select 11. Specifies input for IRQ11. 00 PH11 pin. 01 PJ11 pin. 10 Reserved. 11 Reserved.
ESEL10	External IRQ Input Select 10. Specifies input for IRQ10. 00 PH10 pin. 01 PJ10 pin. 10 PK10 pin. 11 Reserved.
ESEL9	External IRQ Input Select 9. Specifies input for IRQ9. 00 PH9 pin. 01 PJ9 pin. 10 PK9 pin. 11 Reserved.
ESEL8	External IRQ Input Select 8. Specifies input for IRQ8. 00 PH8 pin. 01 PJ8 pin. 10 PK8 pin. 11 Reserved.
ESEL7	External IRQ Input Select 7. Specifies input for IRQ7. 00 PH7 pin. 01 PJ7 pin. 10 PK7 pin. 11 Reserved.

Table 8-32. SIU_ISEL2A Field Descriptions (continued)

Field	Description
ESEL6	External IRQ Input Select 6. Specifies input for IRQ6. 00 PH6 pin. 01 PJ6 pin. 10 PK6 pin. 11 Reserved.
ESEL5	External IRQ Input Select 5. Specifies input for IRQ5. 00 PH5 pin. 01 PJ5 pin. 10 PK5 pin. 11 Reserved.
ESEL4	External IRQ Input Select 4. Specifies input for IRQ4. 00 PH4 pin. 01 PJ4 pin. 10 PK4 pin. 11 Reserved.
ESEL3	External IRQ Input Select 3. Specifies input for IRQ3. 00 PH3 pin. 01 PJ3 pin. 10 PK3 pin. 11 Reserved.
ESEL2	External IRQ Input Select 2. Specifies input for IRQ2. 00 PH2 pin. 01 PJ2 pin. 10 PK2 pin. 11 Reserved.
ESEL1	External IRQ Input Select 1. Specifies input for IRQ1. 00 PH1 pin. 01 PJ1 pin. 10 PK1 pin. 11 Reserved.
ESEL0	External IRQ Input Select 0. Specifies input for IRQ0. 00 PH0 pin. 01 PJ0 pin. 10 PK0 pin. 11 Reserved.

8.3.2.27 Parallel GPIO Pin Data Output Register 0 (SIU_PGPDO0)

The SIU_PGPDO0 register contains the parallel GPIO pin data output for PB[0:15].

Reads and writes to this register are coherent with the registers SIU_GPDO16_19, SIU_GPDO20_23, SIU_GPDO24_27, and SIU_GPDO28_31.

NOTE

On the PXN20, the port A pins are general-purpose inputs only. Therefore, there are no parallel GPIO pin data output register bits for port A.

Offset: SIU_BASE + 0xC00 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	PB0:PB15															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 8-35. Parallel GPIO Pin Data Output Register 0 (SIU_PGPDO0)

8.3.2.28 Parallel GPIO Pin Data Output Register 1 (SIU_PGPDO1)

The SIU_PGPDO1 register contains the parallel GPIO pin data output for PC[0:15] and PD[0:15].

Reads and writes to this register are coherent with the registers SIU_GPDO32_35, SIU_GPDO36_39, SIU_GPDO40_43, SIU_GPDO44_47, SIU_GPDO48_51, SIU_GPDO52_55, SIU_GPDO56_59, and SIU_GPDO60_63.

Offset: SIU_BASE + 0x0C04 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	PC0:PC15															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	PD0:PD15															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 8-36. Parallel GPIO Pin Data Output Register 1 (SIU_PGPDO1)

8.3.2.29 Parallel GPIO Pin Data Output Register 2 (SIU_PGPDO2)

The SIU_PGPDO2 register contains the Parallel GPIO Pin Data Output for PE[0:15] and PF[0:15].

Reads and writes to this register are coherent with the registers SIU_GPDO64_67, SIU_GPDO68_71, SIU_GPDO72_75, SIU_GPDO76_79, SIU_GPDO80_83, SIU_GPDO84_87, SIU_GPDO88_91, and SIU_GPDO92_95.

Offset SIU_BASE + 0x0C08 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	PE0:PE15															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	PF0:PF15															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 8-37. Parallel GPIO Pin Data Output Register 2 (SIU_PGPDO2)

8.3.2.30 Parallel GPIO Pin Data Output Register 3 (SIU_PGPDO3)

The SIU_PGPDO3 register contains the parallel GPIO pin data output for PG[0:15]and PH[0:15].

Reads and writes to this register are coherent with the registers SIU_GPDO96_99, SIU_GPDO100_103, SIU_GPDO104_107, SIU_GPDO108_111, SIU_GPDO112_115, SIU_GPDO116_119, SIU_GPDO120_123, and SIU_GPDO124_127.

Offset: SIU_BASE + 0x0C0C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	PG0:PG15															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	PH0:PH15															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 8-38. Parallel GPIO Pin Data Output Register 3 (SIU_PGPDO3)

8.3.2.31 Parallel GPIO Pin Data Output Register 4 (SIU_PGPDO4)

The SIU_PGPDO4 register contains the parallel GPIO pin data output for PJ[0:15] and PK[0:10].

Reads and writes to this register are coherent with the registers SIU_GPDO18_131, SIU_GPDO132_135, SIU_GPDO136_139, SIU_GPDO140_143, SIU_GPDO144_147, and SIU_GPDO148_151, and SIU_GPDO152_154.

Offset: SIU_BASE + 0x0C10 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	PJ0:PJ15															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	PK0:PK10											0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 8-39. Parallel GPIO Pin Data Output Register 4 (SIU_PGPDO4)

8.3.2.32 Parallel GPIO Pin Data Input Register 0 (SIU_PGPDI0)

Reads to the SIU_PGPDI0 register provide the parallel GPIO pin data input for PA[0:15] and PB[0:15]. Writes have no effect.

Reads of this register are coherent with the registers SIU_GPDI0_3, SIU_GPDI4_7, SIU_GPDI8_11, SIU_GPDI12_15, SIU_GPDI16_19, SIU_GPDI20_23, SIU_GPDI24_27, and SIU_GPDI28_31.

Offset: SIU_BASE + 0x0C40 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	PA0:PA15															
W																
Reset	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	PB0:PB15															
W																
Reset	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U

Figure 8-40. Parallel GPIO Pin Data Input Register 0 (SIU_PGPDI0)

8.3.2.33 Parallel GPIO Pin Data Input Register 1 (SIU_PGPDI1)

Reads to the SIU_PGPDI1 register provide the parallel GPIO pin data input for PC0:PC15 and PD0:PD15. Writes have no effect.

Reads of this register are coherent with the registers SIU_GPDI32_35, SIU_GPDI36_39, SIU_GPDI40_43, SIU_GPDI44_47, SIU_GPDI48_51, SIU_GPDI52_55, SIU_GPDI56_59, and SIU_GPDI60_63.

Offset: SIU_BASE + 0x0C44 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	PC0:PC15															
W																
Reset	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	PD0:PD15															
W																
Reset	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U

Figure 8-41. Parallel GPIO Pin Data Input Register 1 (SIU_PGPDI1)

8.3.2.34 Parallel GPIO Pin Data Input Register 2 (SIU_PGPDI2)

Reads to the SIU_PGPDI2 register provide the parallel GPIO pin data input for PE0:PE15 and PF0:PF15. Writes have no effect.

Reads of this register are coherent with the registers SIU_GPDI64_67, SIU_GPDI68_71, SIU_GPDI72_75, SIU_GPDI76_79, SIU_GPDI80_83, SIU_GPDI84_87, SIU_GPDI88_91, and SIU_GPDI92_95.

Offset: SIU_BASE + 0x0C48 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	PE0:PE15															
W																
Reset	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	PF0:PF15															
W																
Reset	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U

Figure 8-42. Parallel GPIO Pin Data Input Register 2 (SIU_PGPDI2)

8.3.2.35 Parallel GPIO Pin Data Input Register 3 (SIU_PGPDI3)

Reads to the SIU_PGPDI2 register provide the parallel GPIO pin data input for PG0:PG15 and PH0:PH15. Writes have no effect.

Reads of this register are coherent with the registers SIU_GPDI96_99, SIU_GPDI100_103, SIU_GPDI104_107, SIU_GPDI108_111, SIU_GPDI112_115, SIU_GPDI116_119, SIU_GPDI120_123, and SIU_GPDI124_127.

Offset: SIU_BASE + 0x0C4C Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	PG0:PG15															
W																
Reset	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	PH0:PH15															
W																
Reset	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U

Figure 8-43. Parallel GPIO Pin Data Input Register 3 (SIU_PGPDI3)

8.3.2.36 Parallel GPIO Pin Data Input Register 4 (SIU_PGPDI4)

Reads to the SIU_PGPDI4 register provide the parallel GPIO pin data input for PJ0:PJ15 and PK0:PK10. Writes have no effect.

Reads of this register are coherent with the registers SIU_GPDI128_131, SIU_GPDI132_135, SIU_GPDI136_139, SIU_GPDI140_143, SIU_GPDI144_147, SIU_GPDI148_151, and SIU_GPDI152_154.

Offset: SIU_BASE + 0x0C50 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	PJ0:PJ15															
W																
Reset	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	PK0:PK10											0	0	0	0	0
W																
Reset	U	U	U	U	U	U	U	U	U	U	U	0	0	0	0	0

Figure 8-44. Parallel GPIO Pin Data Input Register 4 (SIU_PGPDI4)

8.3.2.37 Masked Parallel GPIO Pin Data Output Register 1 (SIU_MPGPDO1)

The purpose of the masked parallel GPIO pin data output registers is to allow any combination of bits in a 16-bit parallel GPIO pin data output port to be updated in a single 32-bit write operation, while allowing other bits to maintain their previous state. This is accomplished by grouping each 16-bit port with a 16-bit mask register, and only updating those bits in the data register for which the corresponding mask bit is set.

For example, if the current state of the port B parallel GPIO pin data output register is 0x1234 and you want to change only bits [12:15] (i.e., the 4) to be an 8, then a 32-bit write with a mask value of 0x000C and data value of 0x0008 (i.e.,0x000C_0008) would be performed.

The masked parallel GPIO pin data output registers always read as 0.

The SIU_MPGPDO1 register contains the Masked Parallel GPIO Pin Data Output for PB[0:15].

Writes to this register are coherent with the registers SIU_GPDO16_19, SIU_GPDO20_23, SIU_GPDO24_27, and SIU_GPDO28_31.

Offset: SIU_BASE + 0x0C84 Access: User write-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	PB_MASK[0:15]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	PB[0:15]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 8-45. Masked Parallel GPIO Pin Data Output Register 1 (SIU_MPGPDO1)

8.3.2.38 Masked Parallel GPIO Pin Data Output Register 2 (SIU_MPGPDO2)

The SIU_MPGPDO2 register contains the masked parallel GPIO pin data output for PC[0:15].

Writes to this register are coherent with the registers SIU_GPDO32_35, SIU_GPDO36_39, SIU_GPDO40_43, and SIU_GPDO44_47.

Offset: SIU_BASE + 0x0C88 Access: User write-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	PC_MASK[0:15]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	PC[0:15]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 8-46. Masked Parallel GPIO Pin Data Output Register 2 (SIU_MPGPDO2)

8.3.2.39 Masked Parallel GPIO Pin Data Output Register 3 (SIU_MPGPDO3)

The SIU_MPGPDO3 register contains the masked parallel GPIO pin data output for PD[0:15].

Writes to this register are coherent with the registers SIU_GPDO48_51, SIU_GPDO52_55, SIU_GPDO56_59, and SIU_GPDO60_63.

Offset: SIU_BASE + 0x0C8C Access: User write-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	PD_MASK[0:15]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	PD[0:15]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 8-47. Masked Parallel GPIO Pin Data Output Register 3 (SIU_MPGPDO3)

8.3.2.40 Masked Parallel GPIO Pin Data Output Register 4 (SIU_MPGPDO4)

The SIU_MPGPDO4 register contains the masked parallel GPIO pin data output for PE[0:15].

Writes to this register are coherent with registers SIU_GPDO64_67, SIU_GPDO68_71, SIU_GPDO72_75, and SIU_GPDO76_79.

Offset: SIU_BASE + 0x0C90 Access: User write-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	PE_MASK[0:15]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	PE[0:15]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 8-48. Masked Parallel GPIO Pin Data Output Register 4 (SIU_MPGPDO4)

8.3.2.41 Masked Parallel GPIO Pin Data Output Register 5 (SIU_MPGPDO5)

The SIU_MPGPDO5 register contains the masked parallel GPIO pin data output for PF[0:15].

Writes to this register are coherent with registers SIU_GPDO80_83, SIU_GPDO84_87, SIU_GPDO88_91, and SIU_GPDO92_95.

Offset: SIU_BASE + 0x0C94 Access: User write-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	PF_MASK[0:15]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	PF[0:15]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 8-49. Masked Parallel GPIO Pin Data Output Register 5 (SIU_MPGPDO5)

8.3.2.42 Masked Parallel GPIO Pin Data Output Register 6 (SIU_MPGPDO6)

The SIU_MPGPDO6 register contains the masked parallel GPIO pin data output for PG[0:15]

Writes to this register are coherent with registers SIU_GPDO96_99, SIU_GPDO100_103, SIU_GPDO104_107, and SIU_GPDO108_111.

Offset: SIU_BASE + 0x0C98 Access: User write-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	PG_MASK[0:15]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	PG[0:15]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 8-50. Masked Parallel GPIO Pin Data Output Register 6 (SIU_MPGPDO6)

8.3.2.43 Masked Parallel GPIO Pin Data Output Register 7 (SIU_MPGPDO7)

The SIU_MPGPDO7 register contains the masked parallel GPIO pin data output for PH[0:15].

Writes to this register are coherent with registers SIU_GPDO112_115, SIU_GPDO116_119, SIU_GPDO120_123, and SIU_GPDO124_127.

Offset: SIU_BASE + 0xC9C Access: User write-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	PH_MASK[0:15]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	PH[0:15]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 8-51. Masked Parallel GPIO Pin Data Output Register 7 (SIU_MPGPDO7)

8.3.2.44 Masked Parallel GPIO Pin Data Output Register 8 (SIU_MPGPDO8)

The SIU_MPGPDO8 register contains the masked parallel GPIO pin data output for PJ[0:15].

Writes to this register are coherent with registers SIU_GPDO128_131, SIU_GPDO132_135, SIU_GPDO136_139, and SIU_GPDO140_143.

Offset: SIU_BASE + 0x0CA0 Access: User write-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	PJ_MASK[0:15]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	PJ[0:15]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 8-52. Masked Parallel GPIO Pin Data Output Register 8 (SIU_MPGPDO8)

8.3.2.45 Masked Parallel GPIO Pin Data Output Register 9 (SIU_MPGPDO9)

The SIU_MPGPDO8 register contains the masked parallel GPIO pin data output for PK[0:10].

Writes to this register are coherent with registers SIU_GPDO144_147, SIU_GPDO148_151, and SIU_GPDO152_154.

Offset: SIU_BASE + 0x0CA4 Access: User write-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	PK_MASK[0:10]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	PK[0:10]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 8-53. Masked Parallel GPIO Pin Data Output Register 9 (SIU_MPGPDO9)

8.3.2.46 Masked Serial GPO Register for DSPI_A High (SIU_DSPIAH)

The SIU_DSPIAH register allows any combination of bits in the top half of the 32-bit serialized data frame from DSPI_A to be updated with a single 32-bit write operation, while allowing other bits to maintain their previous state. This is accomplished by writing a 16-bit masked value coherently with an update value contained in a 16-bit output field, and only updating those bits in the output register for which the corresponding mask bit is set.

Offset: SIU_BASE + 0x0D00 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	MASK	MASK	MASK	MASK	MASK	MASK	MASK	MASK	MASK	MASK	MASK	MASK	MASK	MASK	MASK	MASK
W	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	DATA	DATA	DATA	DATA	DATA	DATA	DATA	DATA	DATA	DATA	DATA	DATA	DATA	DATA	DATA	DATA
W	31	30	29	28	27	26	25	24	23	222	21	20	19	18	17	16
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 8-54. Masked Serial GPO Register for DSPI_A High (SIU_DSPIAH)

Table 8-33. SIU_DSPIAH Field Descriptions

Field	Description
MASK*n*	Mask Bit. This bit controls the write access to the corresponding GPO for DSPI_A. 0 The previous value defined by GPO for DSPI_A is maintained. 1 The corresponding GPO for DSPI_A is written with the value defined by the DATA*n* field.
DATA*n*	Pin Data Out. This bit stores the data to be driven out on the GPO for DSPI_A output controlled by this register. 0 Logic low value is driven for the corresponding GPO for DSPI_A when this output is selected in the DSPI serialization module. 1 Logic high value is driven for the corresponding GPO for DSPI_A when this output is selected in the DSPI serialization module.

8.3.2.47 Masked Serial GPO Register for DSPI_A Low (SIU_DSPIAL)

The SIU_DSPIAL register allows any combination of bits in the bottom half of the 32-bit serialized data frame from DSPI_A to be updated with a single 32-bit write operation, while allowing other bits to maintain their previous state. This is accomplished by writing a 16-bit masked value coherently with an update value contained in a 16-bit output field, and only updating those bits in the output register for which the corresponding mask bit is set.

Offset: SIU_BASE + 0x0D04 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	MASK 15	MASK 14	MASK 13	MASK 12	MASK 11	MASK 10	MASK 9	MASK 8	MASK 7	MASK 6	MASK 5	MASK 4	MASK 3	MASK 2	MASK 1	MASK 0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	DATA 15	DATA 14	DATA 13	DATA 12	DATA 11	DATA 10	DATA 9	DATA 8	DATA 7	DATA 6	DATA 5	DATA 4	DATA 3	DATA 2	DATA 1	DATA 0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 8-55. Masked Serial GPO Register for DSPI_A Low (SIU_DSPIAL)

Table 8-34. SIU_DSPIAL Field Descriptions

Field	Description
MASK*n*	Mask Bit. This bit controls the write access to the corresponding GPO for DSPI_A. 0 The previous value defined by GPO for DSPI_A is maintained. 1 The corresponding GPO for DSPI_A is written with the value defined by the DATA*n* field.
DATA*n*	Pin Data Out. This bit stores the data to be driven out on the GPO for DSPI_A output controlled by this register. 0 Logic low value is driven for the corresponding GPO for DSPI_A when this output is selected in the DSPI serialization module. 1 Logic high value is driven for the corresponding GPO for DSPI_A when this output is selected in the DSPI serialization module.

8.3.2.48 Masked Serial GPO Register for DSPI_B High (SIU_DSPIBH)

The SIU_DSPIBH register allows any combination of bits in the top half of the 32-bit serialized data frame from DSPI_B to be updated with a single 32-bit write operation, while allowing other bits to maintain their previous state. This is accomplished by writing a 16-bit masked value coherently with an update value contained in a 16-bit output field, and only updating those bits in the output register for which the corresponding mask bit is set.

Offset: SIU_BASE + 0x0D08 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	MASK 31	MASK 30	MASK 29	MASK 28	MASK 27	MASK 26	MASK 25	MASK 24	MASK 23	MASK 22	MASK 21	MASK 20	MASK 19	MASK 18	MASK 17	MASK 16
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	DATA 31	DATA 30	DATA 29	DATA 28	DATA 27	DATA 26	DATA 25	DATA 24	DATA 23	DATA 222	DATA 21	DATA 20	DATA 19	DATA 18	DATA 17	DATA 16
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 8-56. Masked Serial GPO Register for DSPI_B High (SIU_DSPIBH)

Table 8-35. SIU_DSPIBH Field Descriptions

Field	Description
MASK*n*	Mask Bit. This bit controls the write access to the corresponding GPO for DSPI_B. 0 The previous value defined by GPO for DSPI_B is maintained. 1 The corresponding GPO for DSPI_B is written with the value defined by the DATA*n* field.
DATA*n*	Pin Data Out. This bit stores the data to be driven out on the GPO for DSPI_B output controlled by this register. 0 Logic low value is driven for the corresponding GPO for DSPI_B when this output is selected in the DSPI serialization module. 1 Logic high value is driven for the corresponding GPO for DSPI_B when this output is selected in the DSPI serialization module.

8.3.2.49 Masked Serial GPO Register for DSPI_B Low (SIU_DSPIBL)

The SIU_DSPIBL register allows any combination of bits in the bottom half of the 32-bit serialized data frame from DSPI_B to be updated with a single 32-bit write operation, while allowing other bits to maintain their previous state. This is accomplished by writing a 16-bit masked value coherently with an update value contained in a 16-bit output field, and only updating those bits in the output register for which the corresponding mask bit is set.

Offset: SIU_BASE + 0x0D0C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	MASK 15	MASK 14	MASK 13	MASK 12	MASK 11	MASK 10	MASK 9	MASK 8	MASK 7	MASK 6	MASK 5	MASK 4	MASK 3	MASK 2	MASK 1	MASK 0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	DATA 15	DATA 14	DATA 13	DATA 12	DATA 11	DATA 10	DATA 9	DATA 8	DATA 7	DATA 6	DATA 5	DATA 4	DATA 3	DATA 2	DATA 1	DATA 0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 8-57. Masked Serial GPO Register for DSPI_B Low (SIU_DSPIBL)

Table 8-36. SIU_DSPIBL Field Descriptions

Field	Description
MASK*n*	Mask Bit. This bit controls the write access to the corresponding GPO for DSPI_B. 0 The previous value defined by GPO for DSPI_B is maintained. 1 The corresponding GPO for DSPI_B is written with the value defined by the DATA*n* field.
DATA*n*	Pin Data Out. This bit stores the data to be driven out on the GPO for DSPI_B output controlled by this register. 0 Logic low value is driven for the corresponding GPO for DSPI_B when this output is selected in the DSPI serialization module. 1 Logic high value is driven for the corresponding GPO for DSPI_B when this output is selected in the DSPI serialization module.

8.3.2.50 Masked Serial GPO Register for DSPI_C High (SIU_DSPICH)

The SIU_DSPICH register allows any combination of bits in the top half of the 32-bit serialized data frame from DSPI_C to be updated with a single 32-bit write operation, while allowing other bits to maintain their previous state. This is accomplished by writing a 16-bit masked value coherently with an update value contained in a 16-bit output field, and only updating those bits in the output register for which the corresponding mask bit is set.

Offset: SIU_BASE + 0x0D10 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	MASK 31	MASK 30	MASK 29	MASK 28	MASK 27	MASK 26	MASK 25	MASK 24	MASK 23	MASK 22	MASK 21	MASK 20	MASK 19	MASK 18	MASK 17	MASK 16
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	DATA 31	DATA 30	DATA 29	DATA 28	DATA 27	DATA 26	DATA 25	DATA 24	DATA 23	DATA 222	DATA 21	DATA 20	DATA 19	DATA 18	DATA 17	DATA 16
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 8-58. Masked Serial GPO Register for DSPI_C High (SIU_DSPICH)

Table 8-37. SIU_DSPICH Field Descriptions

Field	Description
MASK*n*	Mask Bit. This bit controls the write access to the corresponding GPO for DSPI_C. 0 The previous value defined by GPO for DSPI_C is maintained. 1 The corresponding GPO for DSPI_C is written with the value defined by the DATA*n* field.
DATA*n*	Pin Data Out. This bit stores the data to be driven out on the GPO for DSPI_C output controlled by this register. 0 Logic low value is driven for the corresponding GPO for DSPI_C when this output is selected in the DSPI serialization module. 1 Logic high value is driven for the corresponding GPO for DSPI_C when this output is selected in the DSPI serialization module.

8.3.2.51 Masked Serial GPO Register for DSPI_C Low (SIU_DSPICL)

The SIU_DSPICL register allows any combination of bits in the bottom half of the 32-bit serialized data frame from DSPI_C to be updated with a single 32-bit write operation, while allowing other bits to

maintain their previous state. This is accomplished by writing a 16-bit masked value coherently with an update value contained in a 16-bit output field, and only updating those bits in the output register for which the corresponding mask bit is set.

Offset: SIU_BASE + 0x0D14 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	MASK 15	MASK 14	MASK 13	MASK 12	MASK 11	MASK 10	MASK 9	MASK 8	MASK 7	MASK 6	MASK 5	MASK 4	MASK 3	MASK 2	MASK 1	MASK 0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	DATA 15	DATA 14	DATA 13	DATA 12	DATA 11	DATA 10	DATA 9	DATA 8	DATA 7	DATA 6	DATA 5	DATA 4	DATA 3	DATA 2	DATA 1	DATA 0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 8-59. Masked Serial GPO Register for DSPI_C Low (SIU_DSPICL)

Table 8-38. SIU_DSPICL Field Descriptions

Field	Description
MASK*n*	Mask Bit. This bit controls the write access to the corresponding GPO for DSPI_C. 0 The previous value defined by GPO for DSPI_C is maintained. 1 The corresponding GPO for DSPI_C is written with the value defined by the DATA*n* field.
DATA*n*	Pin Data Out. This bit stores the data to be driven out on the GPO for DSPI_C output controlled by this register. 0 Logic low value is driven for the corresponding GPO for DSPI_C when this output is selected in the DSPI serialization module. 1 Logic high value is driven for the corresponding GPO for DSPI_C when this output is selected in the DSPI serialization module.

8.3.2.52 Masked Serial GPO Register for DSPI_D High (SIU_DSPIDH)

The SIU_DSPIDH register allows any combination of bits in the top half of the 32-bit serialized data frame from DSPI_D to be updated with a single 32-bit write operation, while allowing other bits to maintain their previous state. This is accomplished by writing a 16-bit masked value coherently with an update value contained in a 16-bit output field, and only updating those bits in the output register for which the corresponding mask bit is set.

Offset: SIU_BASE + 0x0D18 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	MASK 31	MASK 30	MASK 29	MASK 28	MASK 27	MASK 26	MASK 25	MASK 24	MASK 23	MASK 22	MASK 21	MASK 20	MASK 19	MASK 18	MASK 17	MASK 16
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	DATA 31	DATA 30	DATA 29	DATA 28	DATA 27	DATA 26	DATA 25	DATA 24	DATA 23	DATA 222	DATA 21	DATA 20	DATA 19	DATA 18	DATA 17	DATA 16
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 8-60. Masked Serial GPO Register for DSPI_D High (SIU_DSPIDH)

Table 8-39. SIU_DSPIDH Field Descriptions

Field	Description
MASK*n*	Mask Bit. This bit controls the write access to the corresponding GPO for DSPI_D. 0 The previous value defined by GPO for DSPI_D is maintained. 1 The corresponding GPO for DSPI_D is written with the value defined by the DATA*n* field.
DATA*n*	Pin Data Out. This bit stores the data to be driven out on the GPO for DSPI_D output controlled by this register. 0 Logic low value is driven for the corresponding GPO for DSPI_D when this output is selected in the DSPI serialization module. 1 Logic high value is driven for the corresponding GPO for DSPI_D when this output is selected in the DSPI serialization module.

8.3.2.53 Masked Serial GPO Register for DSPI_D Low (SIU_DSPIDL)

The SIU_DSPIDL register allows any combination of bits in the bottom half of the 32-bit serialized data frame from DSPI_D to be updated with a single 32-bit write operation, while allowing other bits to maintain their previous state. This is accomplished by writing a 16-bit masked value coherently with an update value contained in a 16-bit output field, and only updating those bits in the output register for which the corresponding mask bit is set.

Offset: SIU_BASE + 0x0D1C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	MASK 15	MASK 14	MASK 13	MASK 12	MASK 11	MASK 10	MASK 9	MASK 8	MASK 7	MASK 6	MASK 5	MASK 4	MASK 3	MASK 2	MASK 1	MASK 0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	DATA 15	DATA 14	DATA 13	DATA 12	DATA 11	DATA 10	DATA 9	DATA 8	DATA 7	DATA 6	DATA 5	DATA 4	DATA 3	DATA 2	DATA 1	DATA 0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 8-61. Masked Serial GPO Register for DSPI_D Low (SIU_DSPIDL)

Table 8-40. SIU_DSPIDL Field Descriptions

Field	Description
MASK*n*	Mask Bit. This bit controls the write access to the corresponding GPO for DSPI_D. 0 The previous value defined by GPO for DSPI_D is maintained. 1 The corresponding GPO for DSPI_D is written with the value defined by the DATA*n* field.
DATA*n*	Pin Data Out. This bit stores the data to be driven out on the GPO for DSPI_D output controlled by this register. 0 Logic low value is driven for the corresponding GPO for DSPI_D when this output is selected in the DSPI serialization module. 1 Logic high value is driven for the corresponding GPO for DSPI_D when this output is selected in the DSPI serialization module.

8.3.2.54 eMIOS Select Register for DSPI_A (SIU_EMIOSA)

The SIU_EMIOSA register selects the output serialized source for the DSPI_A channel.

Offset: SIU_BASE + 0x0D44 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	EMIOS 31	EMIOS 30	EMIOS 29	EMIOS 28	EMIOS 27	EMIOS 26	EMIOS 25	EMIOS 24	EMIOS 23	EMIOS 22	EMIOS 21	EMIOS 20	EMIOS 19	EMIOS 18	EMIOS 17	EMIOS 16
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	EMIOS 15	EMIOS 14	EMIOS 13	EMIOS 12	EMIOS 11	EMIOS 10	EMIOS 9	EMIOS 8	EMIOS 7	EMIOS 6	EMIOS 5	EMIOS 4	EMIOS 3	EMIOS 2	EMIOS 1	EMIOS 0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 8-62. SIU_EMIOSA Select Register for DSPI_A (SIU_EMIOSA)

Table 8-41. SIU_EMIOSA Field Descriptions

Field	Description
EMIOS*n*	eMIOS Channel Enable. 0 This eMIOS channel is not enabled. 1 This eMIOS channel is enabled.

8.3.2.55 SIU_DSPIAH/L Select Register for DSPI_A (SIU_DSPIAHLA)

The SIU_DSPIAHLA register enables the data path from the Masked Serial GPO register for DSPI_A to the equivalent bit position in the DSPI_A channel frame.

Offset: SIU_BASE + 0x0D48 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	DSPIAH 31	DSPIAH 30	DSPIAH 29	DSPIAH 28	DSPIAH 27	DSPIAH 26	DSPIAH 25	DSPIAH 24	DSPIAH 23	DSPIAH 22	DSPIAH 21	DSPIAH 20	DSPIAH 19	DSPIAH 18	DSPIAH 17	DSPIAH 16
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	DSPIAL 15	DSPIAL 14	DSPIAL 13	DSPIAL 12	DSPIAL 11	DSPIAL 10	DSPIAL 9	DSPIAL 8	DSPIAL 7	DSPIAL 6	DSPIAL 5	DSPIAL 4	DSPIAL 3	DSPIAL 2	DSPIAL 1	DSPIAL 0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 8-63. SIU_DSPIAH/L Select Register for DSPI_A (SIU_DSPIAHLA)

Table 8-42. SIU_DSPIAHLA Field Descriptions

Field	Description
DSPIAH*n*	Data Path Enable for DSPI_A High. 0 Data path disabled to DSPI_A High. 1 Data path enabled to DSPI_A High.
DSPIAL*n*	Data Path Enable for DSPI_A Low. 0 Data path disabled to DSPI_A Low. 1 Data path enabled to DSPI_A Low.

8.3.2.56 eMIOS Select Register for DSPI_B (SIU_EMIOSB)

The SIU_EMIOSB register selects the output serialized source for the DSPI_B channel.

Offset: SIU_BASE + 0x0D54 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	EMIOS 31	EMIOS 30	EMIOS 29	EMIOS 28	EMIOS 27	EMIOS 26	EMIOS 25	EMIOS 24	EMIOS 23	EMIOS 22	EMIOS 21	EMIOS 20	EMIOS 19	EMIOS 18	EMIOS 17	EMIOS 16
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	EMIOS 15	EMIOS 14	EMIOS 13	EMIOS 12	EMIOS 11	EMIOS 10	EMIOS 9	EMIOS 8	EMIOS 7	EMIOS 6	EMIOS 5	EMIOS 4	EMIOS 3	EMIOS 2	EMIOS 1	EMIOS 0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 8-64. eMIOS Select Register for DSPI_B (SIU_EMIOSB)

Table 8-43. SIU_EMIOSB Field Descriptions

Field	Description
EMIOS*n*	eMIOS Channel Enable. 0 This eMIOS channel is not enabled. 1 This eMIOS channel is enabled.

8.3.2.57 SIU_DSPIBH/L Select Register for DSPI_B (SIU_DSPIAHLB)

The SIU_DSPIBHLB register enables the data path from the Masked Serial GPO register for DSPI_B to the equivalent bit position in the DSPI_B channel frame.

Offset: SIU_BASE + 0x0D58 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	DSPI BH 31	DSPI BH 30	DSPI BH 29	DSPI BH 28	DSPI BH 27	DSPI BH 26	DSPI BH 25	DSPI BH 24	DSPI BH 23	DSPI BH 22	DSPI BH 21	DSPI BH 20	DSPI BH 19	DSPI BH 18	DSPI BH 17	DSPI BH 16
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	DSPI BL 15	DSPI BL 14	DSPI BL 13	DSPI BL 12	DSPI BL 11	DSPI BL 10	DSPI BL 9	DSPI BL 8	DSPI BL 7	DSPI BL 6	DSPI BL 5	DSPI BL 4	DSPI BL 3	DSPI BL 2	DSPI BL 1	DSPI BL 0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 8-65. SIU_DSPIBH/L Select Register for DSPI_B (SIU_DSPIBHLB)

Table 8-44. SIU_DSPIBHLB Field Descriptions

Field	Description
DSPIBH*n*	Data Path Enable for DSPI_B High. 0 Data path disabled to DSPI_B High. 1 Data path enabled to DSPI_B High.
DSPIBL*n*	Data Path Enable for DSPI_B Low. 0 Data path disabled to DSPI_B Low. 1 Data path enabled to DSPI_B Low.

8.3.2.58 eMIOS Select Register for DSPI_C (SIU_EMIOSC)

The SIU_EMIOSC register selects the output serialized source for the DSPI_C channel.

Offset: SIU_BASE + 0x0D64 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	EMIOS 31	EMIOS 30	EMIOS 29	EMIOS 28	EMIOS 27	EMIOS 26	EMIOS 25	EMIOS 24	EMIOS 23	EMIOS 22	EMIOS 21	EMIOS 20	EMIOS 19	EMIOS 18	EMIOS 17	EMIOS 16
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	EMIOS 15	EMIOS 14	EMIOS 13	EMIOS 12	EMIOS 11	EMIOS 10	EMIOS 9	EMIOS 8	EMIOS 7	EMIOS 6	EMIOS 5	EMIOS 4	EMIOS 3	EMIOS 2	EMIOS 1	EMIOS 0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 8-66. eMIOS Select Register for DSPI_C (SIU_EMIOSC)

Table 8-45. SIU_EMIOSC Field Descriptions

Field	Description
EMIOS*n*	eMIOS Channel Enable. 0 This eMIOS channel is not enabled. 1 This eMIOS channel is enabled.

8.3.2.59 SIU_DSPICH/L Select Register for DSPI_C (SIU_DSPICHLC)

The SIU_DSPICHLC register enables the data path from the Masked Serial GPO register for DSPI_C to the equivalent bit position in the DSPI_C channel frame.

Offset: SIU_BASE + 0x0D68 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	DSPI CH 31	DSPI CH 30	DSPI CH 29	DSPI CH 28	DSPI CH 27	DSPI CH 26	DSPI CH 25	DSPI CH 24	DSPI CH 23	DSPI CH 22	DSPI CH 21	DSPI CH 20	DSPI CH 19	DSPI CH 18	DSPI CH 17	DSPI CH 16
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	DSPI CL 15	DSPI CL 14	DSPI CL 13	DSPI CL 12	DSPI CL 11	DSPI CL 10	DSPI CL 9	DSPI CL 8	DSPI CL 7	DSPI CL 6	DSPI CL 5	DSPI CL 4	DSPI CL 3	DSPI CL 2	DSPI CL 1	DSPI CL 0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 8-67. SIU_DSPICH/L Select Register for DSPI_C (SIU_DSPICHLC)

Table 8-46. SIU_DSPICHLC Field Descriptions

Field	Description
DSPICH*n*	Data Path Enable for DSPI_C High. 0 Data path disabled to DSPI_C High. 1 Data path enabled to DSPI_C High.
DSPICL*n*	Data Path Enable for DSPI_C Low. 0 Data path disabled to DSPI_C Low. 1 Data path enabled to DSPI_C Low.

8.3.2.60 eMIOS Select Register for DSPI_D (SIU_EMIOSD)

The SIU_EMIOSD register selects the output serialized source for the DSPI_D channel.

Offset: SIU_BASE + 0x0D74 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	EMIOS 31	EMIOS 30	EMIOS 29	EMIOS 28	EMIOS 27	EMIOS 26	EMIOS 25	EMIOS 24	EMIOS 23	EMIOS 22	EMIOS 21	EMIOS 20	EMIOS 19	EMIOS 18	EMIOS 17	EMIOS 16
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	EMIOS 15	EMIOS 14	EMIOS 13	EMIOS 12	EMIOS 11	EMIOS 10	EMIOS 9	EMIOS 8	EMIOS 7	EMIOS 6	EMIOS 5	EMIOS 4	EMIOS 3	EMIOS 2	EMIOS 1	EMIOS 0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 8-68. eMIOS Select Register for DSPI_D (SIU_EMIOSD)

Table 8-47. SIU_EMIOSD Field Descriptions

Field	Description
EMIOS*n*	eMIOS Channel Enable. 0 This eMIOS channel is not enabled. 1 This eMIOS channel is enabled.

8.3.2.61 SIU_DSPIDH/L Select Register for DSPI_D (SIU_DSPIDHLD)

The SIU_DSPIDHLD register enables the data path from the Masked Serial GPO register for DSPI_D to the equivalent bit position in the DSPI_D channel frame.

Offset: SIU_BASE + 0x0D78 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	DSPIDH 31	DSPIDH 30	DSPIDH 29	DSPIDH 28	DSPIDH 27	DSPIDH 26	DSPIDH 25	DSPIDH 24	DSPIDH 23	DSPIDH 22	DSPIDH 21	DSPIDH 20	DSPIDH 19	DSPIDH 18	DSPIDH 17	DSPIDH 16
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	DSPIDL 15	DSPIDL 14	DSPIDL 13	DSPIDL 12	DSPIDL 11	DSPIDL 10	DSPIDL 9	DSPIDL 8	DSPIDL 7	DSPIDL 6	DSPIDL 5	DSPIDL 4	DSPIDL 3	DSPIDL 2	DSPIDL 1	DSPIDL 0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 8-69. SIU_DSPIDH/L Select Register for DSPI_D (SIU_DSPIDHLD)

Table 8-48. SIU_DSPIDHLD Field Descriptions

Field	Description
DSPIDH*n*	Data Path Enable for DSPI_D High. 0 Data path disabled to DSPI_D High. 1 Data path enabled to DSPI_D High.
DSPIDL*n*	Data Path Enable for DSPI_D Low. 0 Data path disabled to DSPI_D Low. 1 Data path enabled to DSPI_D Low.

8.4 Functional Description

The following sections provide an overview of the SIU operation.

8.4.1 System Configuration

8.4.1.1 Boot Configuration

During the assertion of $\overline{\text{RESET}}$, the BOOTCFG pin is used to load a value into the SIU_RSR[BOOTCFG] bit, so the BAM program can determine the location of the reset configuration half word (RCHW), the boot mode to be initiated, and whether to initiate a CAN or SCI boot. See Section 9.3.3.1.1, Reset Configuration Halfword Read, of the BAM chapter for detail on the RCHW. Table 8-49 defines the boot modes specified by the SIU_RST[BOOTCFG] field.

Table 8-49. SIU_RSR[BOOTCFG] Configuration

Value	Meaning
0b0	Boot from internal flash memory
0b1	CAN/SCI boot

8.4.1.2 Pad Configuration

The pad configuration registers (SIU_PCR) in the SIU allow software control of the static electrical characteristics of external pins. The SIU_PCRs can select the multiplexed function of a pin, selection of pullup or pulldown devices, the slew rate of I/O signals, open drain mode for output pins, and hysteresis.

8.4.2 Reset Control

The reset controller logic is located in the SIU. See Section 4.4, Reset Configuration, for reset operation details.

8.4.3 External Interrupt

There are 16 external interrupt inputs, IRQ0–IRQ15, to the SIU. The IRQ*n* inputs can be configured for rising- or falling-edge events or both. Each IRQ*n* input has a corresponding flag bit in the external interrupt status register (SIU_EISR). The flag bits for the IRQ4–IRQ15 inputs are ORed together to form one interrupt request to the interrupt controller. The flag bits for the IRQ1–IRQ0 inputs can generate an interrupt request to the interrupt controller or a DMA transfer request to the DMA controller. Figure 8-70 shows the DMA and interrupt request connections to the interrupt and DMA controllers.

Any pin used as an external interrupt must be configured in its SIU_PCR as a GPIO in input mode. In addition, either rising and/or falling edge must be enabled in the SIU_IREER, or SIU_IFEER.

Two external inputs from pins PC6 and PC5 connect through the SIU to the critical interrupt input to the Z0 and Z6 cores, respectively. These signals should be used as non-maskable interrupt (NMI) inputs.

The SIU contains an overrun interrupt enable for each IRQ and one combined overrun interrupt request to the interrupt controller which is the logical OR of the individual overrun requests' flags. Only the

combined overrun interrupt request is used in the device, and the individual overrun requests are not connected.

Each IRQ pin has a programmable filter for rejecting glitches on the IRQ signals. The filter length for the IRQ pins is specified in the external IRQ digital filter register (SIU_IDFR).

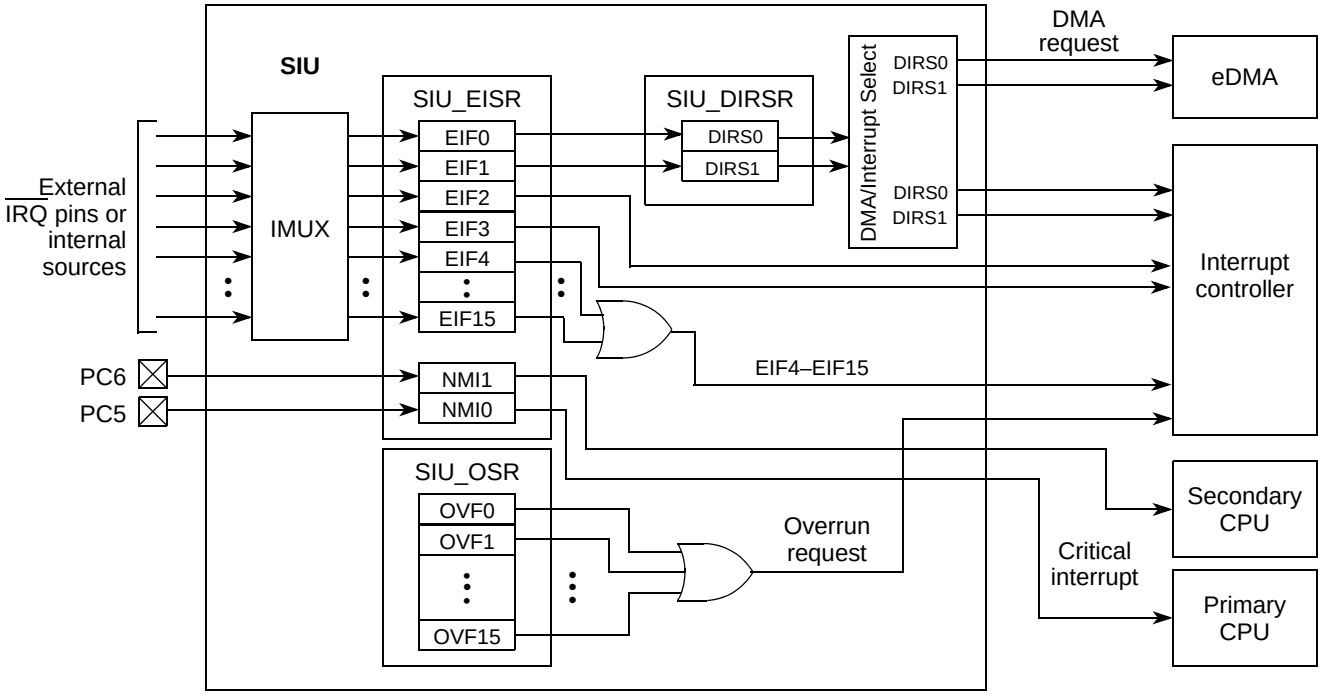


Figure 8-70. SIU DMA/Interrupt Request Diagram

8.4.4 GPIO Operation

All GPIO functionality is provided by the SIU. Each pin that has GPIO functionality has an associated Pin Configuration Register in the SIU where the GPIO function is selected for the pin. In addition, each pin with GPIO functionality has an input data register (SIU_GPDI*n*) and an output data register (SIU_GPDO*n*). The SIU also implements several parallel GPIO registers (SIU_PGPDO*n* and SIU_PGPDI*n*) that can be used to access as many as 32 GPIO bits in single- and word-sized accesses. The values read/written to these parallel register is coherent with the data read/written to the SIU_GPDO*n* and SIU_GPDI*n* registers.

8.4.5 Internal Multiplexing

The IMUX Select Registers (SIU_ISEL*n*) provide selection of the input source for the ADC external trigger inputs and the SIU external interrupts.

8.4.5.1 ADC External Trigger Input Multiplexing

The two ADC external trigger inputs (start of conversion and injected trigger) can be connected to four different external pins or to PIT2. The input source for each ADC external trigger is individually specified in the IMUX Select Register 4 (SIU_ISEL4). Figure Figure 8-71 gives an example of the multiplexing of

an ADC external trigger input. As shown in the figure, the start of conversion input of the ADC can be connected to the PE10 pin, the PE11 pin, PE12 pin, PE13 pin, or the PIT2 channel. The external injected trigger are multiplexed in the same manner.

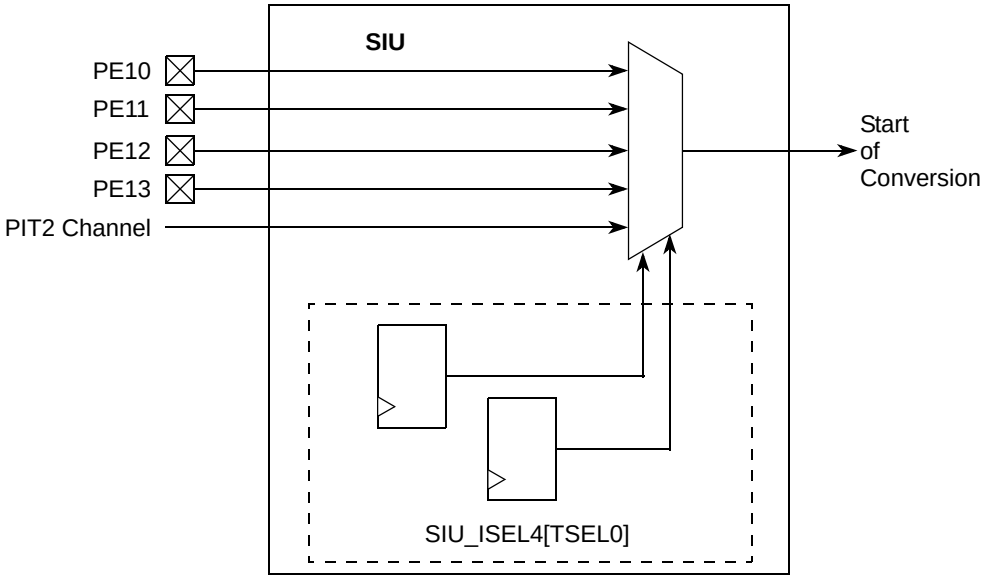


Figure 8-71. Internal Multiplexing Block Diagram

8.4.5.2 SIU External Interrupt Input Multiplexing

The 16 SIU external interrupt inputs can be connected to one of nine external pins. The input source for each SIU external interrupt is individually specified in the IMUX Select Register 1 (SIU_ISEL1), IMUX Select Register 2 (SIU_ISEL2) and IMUX Select Register 2A (SIU_ISEL2A). Figure 8-72 shows an example of the multiplexing of an SIU external interrupt input. As shown in the figure, the IRQ[0] input of the SIU can be connected to the PB0 pin, PC0 pin, PD0 pin, PE0 pin, PF0 pin, PG0 pin, PH0 pin, PJ0 pin, or PK0 pin The remaining IRQ inputs are multiplexed in the same manner.

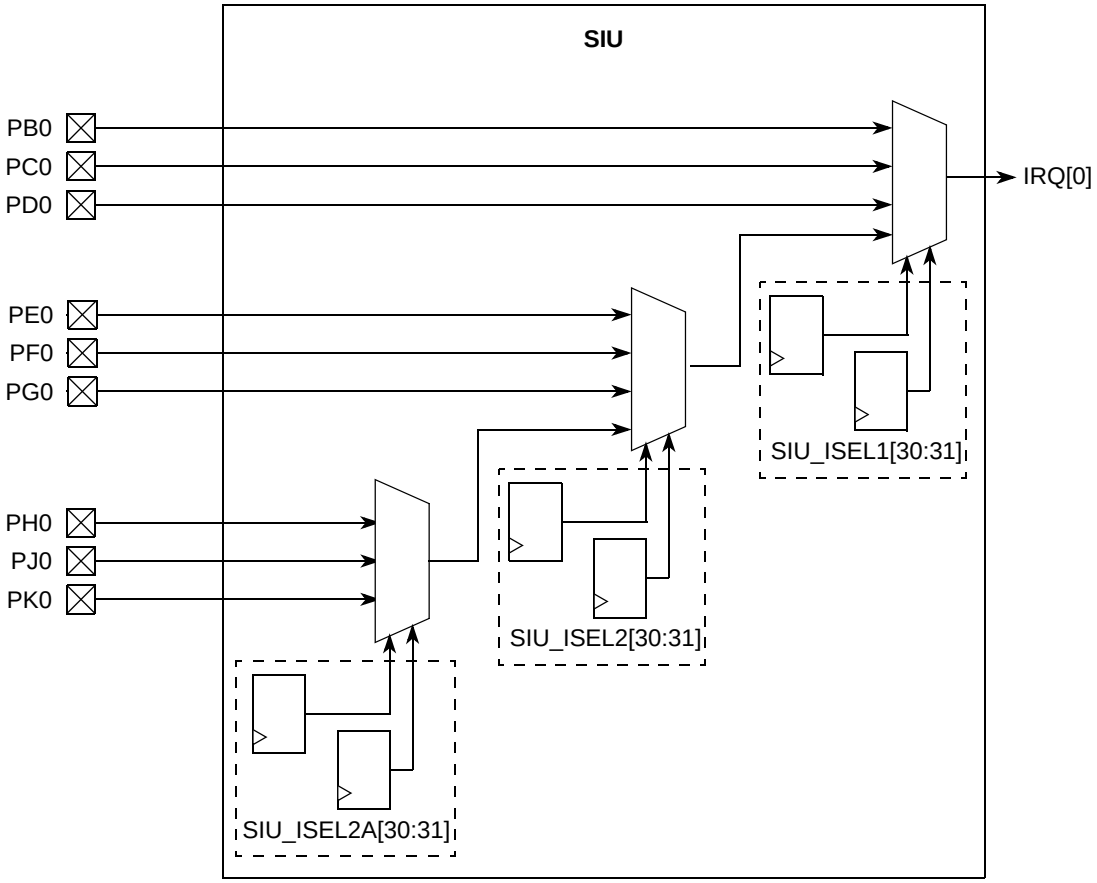


Figure 8-72. SIU External Interrupt Input Multiplexing

8.4.5.3 SIU EMIOS/DSPI Multiplexing

The Serialization Data Register from each of the four DSPI modules can be connected to the EMIOS channel outputs (if selected by the SIU_EMIOSx registers) or the Masked Serial GPO registers (if selected by the SIU_DSPIxHLx registers, as shown in Figure 8-73.

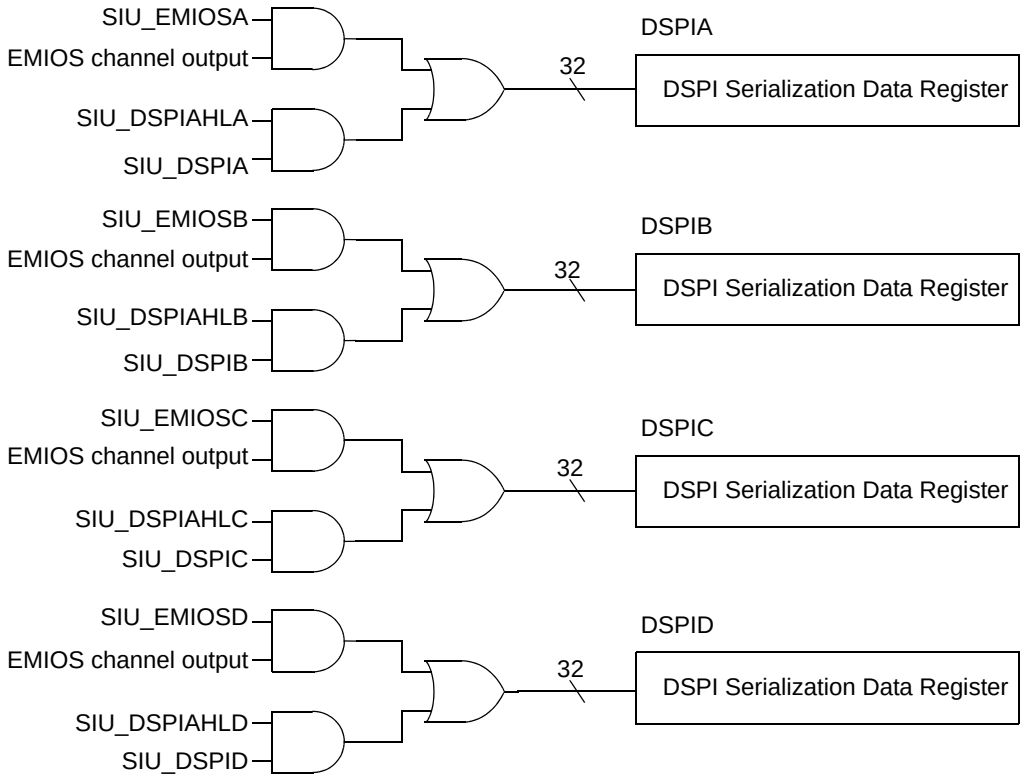


Figure 8-73. SIU EMIOS/DSPI Multiplexing

Chapter 9
Boot Assist Module (BAM)

9.1 Introduction

The PXN20 boot assist module (BAM) is a 4 KB block of read-only memory (ROM) that is programmed by Freescale using variable length encoding (VLE) code. The BAM program is executed by the e200z6 when the PXN20 performs a power-on-reset (POR) or any other reset for which the CRP_Z6VEC register remains in its reset state. The BAM program provides MCU initialization. It also transitions to the user application code that resides in the internal flash or downloads the user code into internal RAM via CAN or SCI serial links and passes control to the user code.

9.1.1 Features

The BAM program provides the following functionality:

- Initial e200z6 core MMU setup with no address translation for all internal MCU resources and external memory address space
- Location and detection of user boot code in the internal flash
- Automatic switch to serial-boot mode if internal flash is blank or invalid
- Supports user programmable 64-bit password protection for serial-boot mode
- Supports serial bootloading via CAN bus or eSCI to the internal SRAM
- Supports censorship protection for internal flash memory
- Provides an option to disable the error correction status module (ECSM) software watchdog timer (enabled by default)
- Configures MMU boot block to boot as either classic Power Book E code or as Freescale VLE code, allowing transition to the user code with classic Power Book E instructions or VLE instructions

NOTE

The BAM program is intended to be run on e200z6 (main core) only. Attempting to execute the BAM program by the e200z0 core may cause erratic MCU behavior.

NOTE

During BAM program execution, the default reset values of various system registers (e.g. SIU, FlexCAN, eSCI, ECSM, SWT) may be updated from their default reset values.

9.1.2 Modes of Operation

The BAM has the following modes of operation:

- Normal mode
- Debug mode
- Internal-boot mode
- Serial-boot mode

9.1.2.1 Normal Mode

In normal operation the BAM responds to all read requests within its address space. The e200z6 core executes the BAM program after the negation of $\overline{\text{RESET}}$ if the CRP_Z6VEC register value is 0xFFFF_FFFC.

9.1.2.2 Debug Mode

The BAM program is not executed when the MCU comes out of reset in OnCE debug mode. The development tool must initialize the MCU instead of BAM before starting the user application.

9.1.2.3 Internal-Boot Mode

This mode of operation is for systems that boot from internal memory. The internal flash is used for all code and all boot configuration data. After the BAM has completed the boot process, user code may enable the external bus interface if required.

9.1.2.4 Serial-Boot Mode

This mode of operation can load a user program into system RAM using the eSCI or FlexCAN serial interface, then execute the downloaded program. The user program can then be used to control the download of data and erasing/programming of the internal or external flash memory.

9.2 Memory Map and Registers

This section provides a detailed description of the BAM memory map.

9.2.1 Module Memory Map

Table 9-1 shows the BAM memory map. The BAM ROM module occupies the last 16 KB of the MCU memory space; however, only 4 KB is physically present. The upper 4 KB shadows the lower 4 KB. Some important absolute addresses are presented in Table 9-1.

Table 9-1. BAM Absolute Addresses

Address	Comment
0xFFFF_FFFC	BAM reset vector—first executed address after the reset
0xFFFF_F000	BAM start address

9.2.2 Register Descriptions

The BAM module does not have any registers.

9.3 Functional Description

9.3.1 BAM Program Resources

The BAM program uses/initializes these MCU resources:

- The BOOTCFG field in the reset status register (SIU_RSR) determines the boot option
- The location and value of the reset configuration halfword (RCHW) determines the location of boot code and the boot configuration options
- FlexCAN_A and eSCI _A modules for serial-boot mode
- FlexCAN_A message buffer RAM for stack and global variables during serial-boot mode
- The DISNEX bit in the SIU_CCR register to determine if the Nexus port is enabled

The BAM program:

- Configures the e200z6 MMU to cover all available MCU address space: internal flash, peripheral bridge, and SRAM without address translation
- Configures FlexCAN_A, eSCI_A when performing serial-boot mode
- Uses the eDMA during serial-boot mode.

9.3.2 BAM Program Operation

If the CRP_Z6VEC register value is 0xFFFF_FFFC, the BAM code is executed after the negation of reset and before user code starts. To prevent the execution of the BAM code when exiting sleep mode, change the value of the CRP_Z6VEC register before entering sleep mode. See Chapter 6, Clocks, Reset, and Power (CRP), for more detail about the CRP_Z6VEC register.

The BAM reads the status of the BOOTCFG bit from the reset status register (SIU_RSR) and the appropriate boot sequence is started (see Table 9-2).

Table 9-2 shows the encoding of the BOOTCFG bit in the SIU_RSR, with the value stored in the Censorship word in the shadow row of internal flash memory. The table also shows whether the internal flash memory is enabled or disabled, whether the Nexus port is enabled or disabled, and whether the password downloaded in serial-boot mode is compared with a fixed public password or compared to a user programmable flash password.

Table 9-2. Boot Modes

Boot Mode Name	BOOTCFG	Censorship Control 0x00FF_FDE0	Serial Boot Control 0x00FF_FDE2	Internal Flash State	Nexus State	Serial Password
Internal—Censored	0	Any other value	Don't care	Enabled	Disabled	Flash
Internal—Public		0x55AA		Enabled	Enabled	Public

Table 9-2. Boot Modes (continued)

Boot Mode Name	BOOTCFG	Censorship Control 0x00FF_FDE0	Serial Boot Control 0x00FF_FDE2	Internal Flash State	Nexus State	Serial Password
Serial—Flash Password	1	Don't care	0x55AA	Enabled	Disabled	Flash
Serial—Public Password			Any other value	Disabled	Enabled	Public

The censorship control word (CCW) is a 32-bit word of data stored in the shadow row of internal flash memory. This memory location is read and interpreted by hardware as part of the boot process. It is used with the BOOTCFG bit to enable/disable the internal flash memory and the Nexus interface. The memory address of the censorship control word is 0x00FF_FDE0. The censorship control word is programmed to be 0x55AA_55AA. This results in a device that is not censored and uses a flash-based password for serial-boot mode.

Censorship control word at 0x00FF_FDE0:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	0	1	0	1	0	1	1	0	1	0	1	0	1	0
Censorship control—showing an uncensored part (factory default 0x55AA)															

16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	1	0	1	0	1	0	1	1	0	1	0	1	0	1	0
Serial boot control—showing the use of the flash-based password (factory default 0x55AA)															

Figure 9-1. Censorship Control Word (CCW)

The BAM code uses the state of the DISNEX bit to determine if the serial password downloaded in serial-boot mode is compared to a fixed public value (0xFEED_FACE_CAFE_BEEF) or is compared to a flash value stored in the shadow row of internal flash at address 0x00FF_FDD8.

Serial-boot flash password at 0x00FF_FDD8 – 0x00FF_FDDF:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	1	1	1	1	1	0	1	1	1	0	1	1	0	1
Serial boot password (0x00FF_FDD8)–0xFEED (factory default)															

16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
1	1	1	1	1	0	1	0	1	1	0	0	1	1	1	0
Serial boot password (0x00FF_FDDA)–0xFACE (factory default)															

32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
1	1	0	0	1	0	1	0	1	1	1	1	1	1	1	0
Serial boot password (0x00FF_FDDC)–0xCAFE (factory default)															

48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
1	0	1	1	1	1	1	0	1	1	1	0	1	1	1	1
Serial boot password (0x00FF_FDDE)– 0xBEEF (factory default)															

Figure 9-2. Serial Boot Flash Password

If the BAM fails to find a valid RCHW in internal-boot mode then serial-boot mode is entered. Therefore a serial password must be provided for both values of the BOOTCFG bit.

9.3.3 Features

Because the MMU default out of reset is to allow access to the 4 KB range around the reset vector only, the BAM program sets up the e200z6 core MMU to enable accesses to all MCU resources, as described in Table 9-3.

Table 9-3. MMU Configuration for an Internal Boot

TLB Entry	Region	Logical Base Address	Physical Base Address	Size	Attributes
0	Peripheral bridge and BAM	0xFFF0_0000	0xFFF0_0000	1 MB	Cache inhibited Guarded Big Endian Global PID
1	Internal flash	0x0000_0000	0x0000_0000	256 MB	Cache enabled Not guarded Big Endian Global PID

Table 9-3. MMU Configuration for an Internal Boot

TLB Entry	Region	Logical Base Address	Physical Base Address	Size	Attributes
2	reserved[1]	0x2000_0000	0x2000_0000	256 MB	Cache enabled Not guarded Big Endian Global PID
3	SRAM	0x4000_0000	0x4000_0000	256 KB	Cache inhibited Not guarded Big Endian Global PID

1 The MMU can be programmed at this address range, but nothing responds to an access.

Code type attributes for TLB entries 1–3 are set according to the coding of the user application (VLE or classic Power Book E).

After configuring the MMU, the BAM determines the selected boot mode and provides the following features for each of the boot modes:

9.3.3.1 Internal-Boot Mode

When the core determines that internal-boot mode has been selected, a machine check exception is configured to handle possible ECC read errors that may occur while searching the internal flash to find the reset configuration halfword (RCHW).

9.3.3.1.1 Reset Configuration Halfword Read

The BAM searches the internal flash memory for a valid RCHW. A valid RCHW is a 16-bit value that contains a fixed 8-bit boot identifier and some configuration bits. The RCHW is expected to be the first halfword in one of the low-address space small flash blocks.

The memory addresses of the six locations searched for a valid RCHW are shown in Table 9-4.

Table 9-4. LAS Block Memory Addresses

Block	Address
0	0x0000_0000
1	0x0000_4000
4	0x0001_0000
7	0x0001_C000
8	0x0002_0000
9	0x0003_0000

The BOOT_BLOCK_ADDRESS used in the register descriptions below is the address in Table 9-5 where the BAM finds a valid RCHW.

Figure 9-3 shows the fields of the RCHW.

Offset: BOOT_BLOCK_ADDRESS + 0x0000 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	WTE	0	VLE	BOOT ID							
W																
Reset	0	0	0	0	0	0	0	0	0	1	0	1	1	0	1	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 9-3. RCHW Fields

Table 9-5. Internal Boot RCHW Field Descriptions

Field	Description
WTE	Watchdog timer enable. This bit determines if the software watchdog timer is disabled. 0 Disable software watchdog timer 1 Software watchdog timer maintains its default state out of reset, (i.e.,enabled).
VLE	VLE Code Indicator. This bit is used to configure the MMU to execute the user code as either Classic Book E code or as Freescale VLE code. 0 User code executes as classic Book E code. 1 User code executes as Freescale VLE code.
BOOTID	Boot identifier. This field serves two functions. First, it is used to indicate which block in flash memory contains the boot program. Second, it identifies whether the flash memory is programmed or invalid. The value of a valid boot identifier is 0x5A (0b01011010). The BAM program checks the first halfword of each flash memory block starting at block 0 until a valid boot identifier is found. If all blocks in the low- address space of the internal flash are checked and no valid boot identifier is found, the internal flash is assumed to be invalid and a CAN/SCI boot is initiated.

If the BAM fails to find a valid RCHW, it assumes the flash is erased or corrupt and switches to serial-boot mode.

If the BAM finds a valid RCHW, the configuration bits are parsed as shown in Table 9-4. The BAM then fetches the reset vector from the address of the RCHW + 4, and program execution continues from that address.

Offset: BOOT_BLOCK_ADDRESS + 0x0004 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12	A13	A14	A15
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	A16	A17	A18	A19	A20	A21	A22	A23	A24	A25	A26	A27	A28	A29	A30	A31
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 9-4. Reset Boot Vector

9.3.3.2 Serial-Boot Mode Features

In this mode of operation, the BAM code configures FlexCAN_A and eSCI_A for serial download of a user program. Unused message buffers in FlexCAN_A are used for stack and global variables. The system clock is selected directly from main crystal oscillator output; thus, the crystal frequency defines baud rates for serial interfaces, used to download the user application.

9.3.3.2.1 FlexCAN and eSCI Configuration

The BAM program configures FlexCAN_A and eSCI_A to receive messages. The CNRX_A and the RXD_A pads are configured as inputs to the FlexCAN and eSCI modules. The CNTX_A pad is configured as an output from the FlexCAN module.The TXD_A pad remains configured as GPIO input until a valid eSCI byte is received before a valid CAN message.

The FlexCAN controller is configured to operate at a baud rate = system clock frequency / 40, using the standard 11 bit identifier format detailed in CAN 2.0A specification. (See Table 9-6 for examples of baud rates.)

The BAM program ignores all possible errors that may happen during the serial communication. All received data is assumed to be good and is echoed on the CNTX signal.

The CAN controller bit timing is programmed with 10 time quantas and the sample point is two time quantas before the end (see Figure 9-5).

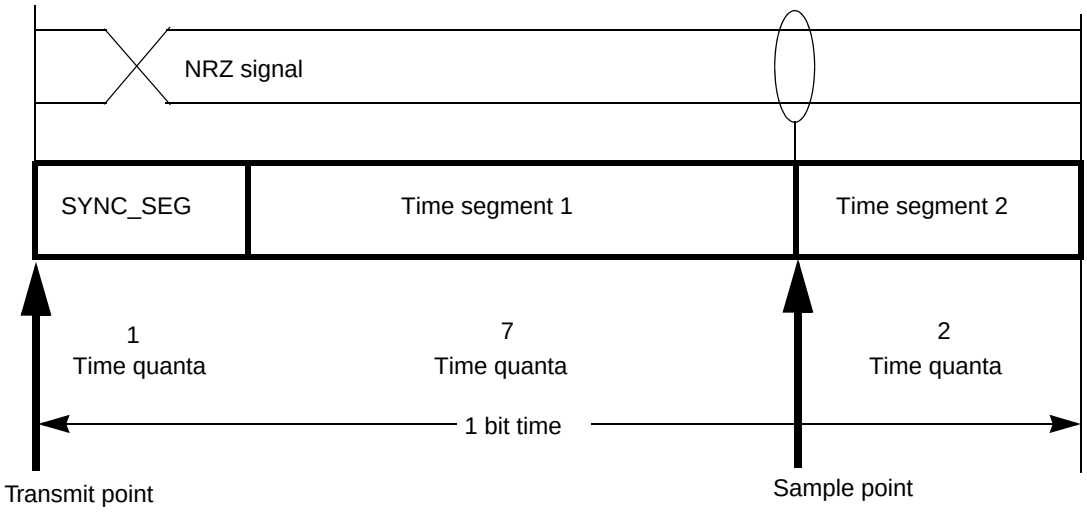


Figure 9-5. FlexCAN Bit Timing

The eSCI is configured for one start bit, eight data bits, no parity, and one stop bit. It operates at a baud rate = system clock / 832. (See Table 9-6 for baud rate examples.)

NOTE

The BAM program during the serial download routine reconfigures the software watchdog timer timeout (SWT_TO) from a default of 0x2_7100 to 0x800_0000. The SWT is always clocked from the 16 MHz IRC.

The BAM program ignores eSCI errors. All received data is assumed to be good and is echoed out on the TXD_A signal.

Table 9-6. Serial-Boot Mode—Baud Rates

Crystal Frequency (MHz)	SCI Baud Rate (baud)	CAN Baud Rate (baud)
f_{extal}	f_{extal} / 832	f_{extal} / 40
8	9600	200K
12	14400	300K
16	19200	400K
20	24000	500K
40	48000	1M

Upon reception of a valid CAN message with ID = 0x011 that contains 8 bytes of data, or an eSCI byte, the BAM program transitions to one of two serial boot sub-modes: CAN serial-boot mode or eSCI serial-boot mode.

In CAN serial-boot mode, the eSCI_A RXD_A pad reverts to GPIO input. The ensuing download protocol is assumed to be all through the CAN, eSCI is disabled.

If the eSCI byte is received first, the CAN_A controller is disabled and its pad reprogrammed to the GPIO, the TXD_A signal is reconfigured as an output. CAN messages are ignored.

Table 9-7. CAN/eSCI Reset Configuration for CAN/eSCI Pins in Serial-Boot Mode

Pins	Reset Function	Initial Serial-Boot Mode	Serial-Boot Mode After a Valid CAN Message Received	Serial-Boot Mode After a Valid eSCI Message Received
CNTX_A	GPIO	CNTX_A	CNTX_A	GPIO
CNRX_A	GPIO	CNRX_A	CNRX_A	GPIO
TXD_A	GPIO	GPIO	GPIO	TXD_A
RXD_A	GPIO	RXD_A	GPIO	RXD_A

Table 9-8. CAN/eSCI Reset Pin Configuration

Pins	I/O	Hysteresis	Driver Configuration
CNTX_A / TXD_A	Output	—	Push/Pull
CNRX_A / RXD_A	Input	Y	—

9.3.3.2.2 Serial-Boot Mode Download Protocol

The download protocol follows four steps:

1. Send 64-bit password.
2. Send start address, size of download code in bytes and VLE bit.
3. Download data.
4. Switch to the loaded code at start address.

The communication is done in half-duplex manner, any transmission from host is followed by the MCU transmission. The host computer will not send data until it receives echo from the MCU. All multibyte data structures are sent most significant byte (MSB) first.

When the CAN is used for serial download, the data is packed into standard CAN messages in the following manner:

- A message with 0x0011 ID and 8-byte length is used to send the password. The MCU echoes with the same data, but ID = 0x0001.
- A message with 0x0012 ID and 8-byte length is used to send the start address, length, and the VLE mode bit. The MCU echoes with a message with 0x002 ID.
- Messages with 0x0013 ID are used to send the downloaded data. The MCU echoes with 0x003 ID.

When the SCI is used for serial download, the data is sent byte-by-byte.

9.3.3.2.3 Serial-Boot Mode Processing

The BAM program executes the serial boot as following:

1. Download 64-bit password.

   The received 8-byte password is checked for validity. It is checked to ensure that none of the 4 × 16-bit halfwords are illegal passwords, such as 0x0000 or 0xFFFF. A password must have at least one 0 and one 1 in each halfword lane to be considered legal.

   The BAM program then checks the censorship status of the MCU by checking the DISNEX bit in the SIU_CCR register. If Nexus is disabled, the MCU is considered to be censored and the password is compared with a password stored in the shadow row in internal flash memory.

   If Nexus is enabled, the MCU is not considered to be censored and the password is compared to the fixed value = 0xFEED_FACE_CAFE_BEEF.

   If the password fails any validity test, the MCU stops responding to all stimulus. Then the RESET signal must be asserted or the software watchdog must be allowed to time out. If the password is valid, the BAM refreshes the software watchdog timer and performs the next step in the protocol.

2. Download start address, size of download, and VLE bit.

   The next 8 bytes received by the MCU are considered to contain a 32-bit start address, the VLE mode bit, and a 31-bit code length (see Figure 9-6).

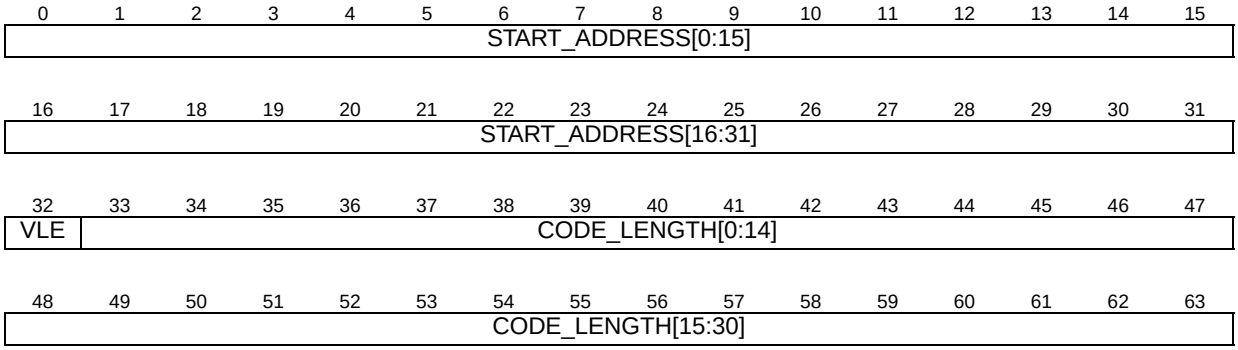


Figure 9-6. Start Address, VLE Bit and Download Size in Bytes

   The start address defines where the received data is stored and where the MCU branches after the download is finished. The two least significant bits of the start address are ignored by the BAM program, such that the loaded code should be 32-bit word aligned.

   The length defines how many data bytes are loaded.

   The VLE mode bit instructs the MCU to program MMU pages with VLE attribute. If it is 1, the downloaded code must be compiled to VLE instructions. If it is 0, the code contains classic Power Book E architecture instructions.

3. Download data.

   Each byte of data received is stored in the MCU's memory, starting at the address specified in the previous protocol step and incrementing through memory until the number of bytes of data received and stored in memory matches the number specified in the previous protocol step.

NOTE

In the PXN20, the SRAM is protected by 64-bit wide error correction code (ECC). In the general case, this means any write to uninitialized SRAM must be 64 bits wide, otherwise an ECC error may occur. Therefore the BAM buffers downloaded data until 8 bytes have been received, and then does a single 64-bit wide write. Only system RAM supports 64-bit writes; therefore, attempting to download data to other RAM apart from system RAM will cause errors. If the start address of the downloaded data is not on an 8-byte boundary, the BAM will write 0x0000 to the memory locations from the proceeding 8-byte boundary to the start address (maximum 4 bytes). The BAM will also write 0x0000 to all memory locations from the last byte of data downloaded to the following 8 byte boundary (maximum 7 bytes).

4. Switch to the loaded code.

   The BAM program waits for the last echo message transmission to complete, then the active communication controller is disabled. Its pins revert to GPIO inputs. The BAM code passes control to the loaded code at start address, which was received in step 2 of the protocol.

NOTE

The code that is downloaded and executed must periodically refresh the platform watchdog timer or change the timeout period to a value that will not cause resets during normal operation.

The serial download protocol is summarized in Table 9-9 and Table 9-10.

Table 9-9. CAN Serial-Boot Mode Download Protocol

Protocol Step	Host Sent Message	BAM Response Message	Action
1	CAN ID 0x0011 + 64-bit password	CAN ID 0x0001 + 64-bit password	Password checked for validity and compared against stored password. Platform watchdog timer is refreshed if the password check is successful.
2	CAN ID 0x0012 + 32-bit store address + VLE bit + 31-bit number of bytes	CAN ID 0x0002 + 32-bit store address + VLE bit + 31-bit number of bytes	Load address and size of download are stored for future use. The VLE bit determines whether the MMU entry for the SRAM, EBI, and flash is configured to run Book E or VLE code.
3	CAN ID 0x0013 + 8 to 64 bits of raw binary data	CAN ID 0x0003 + 8 to 64 bits of raw binary data	Each byte of data received is stored in MCU memory, starting at the address specified in the previous step and incrementing until the amount of data received and stored, matches the size as specified in the previous step.
4	None	None	The BAM returns IO pins to their reset state, disables FlexCAN_A module and then branches to the first address the data was stored to (As specified in step 2).

Table 9-10. eSCI Serial-Boot Mode Download Protocol

Protocol Step	Host Sent Message	BAM Response Message	Action
1	64-bit password MSB first	64-bit password	Password checked for validity and compared against stored password. Platform Watchdog timer is refreshed if the password check is successful.
2	32-bit store address + VLE bit + 31-bit number of bytes (MSB first)	32-bit store address + VLE bit + 31-bit number of bytes	Load address and size of download are stored for future use. The VLE bit determines whether the MMU entry for the SRAM, EBI and Flash is configured to run Book E or VLE code.
3	8 bits of raw binary data	8 bits of raw binary data	Each byte of data received is store in MCU memory, starting at the address specified in the previous step and incrementing until the amount of data received and stored, matched the size as specified in the previous step.
4	None	None	The BAM returns IO pins to their reset state and disables the ESCI_A module. Then it branches to the first address the data was stored to (as specified in step 2).

NOTE

The BAM writes additional two zero double words to the system RAM after loaded code to prevent possible ECC errors, which could happen because the CPU speculatively pre-fetches data after the last loaded instruction, where the RAM may not be initialized.

NOTE

The last loaded code address must not exceed 0x4003_FFF0 (the upper allowed RAM address by MMU settings, minus two zero double words, written by BAM at the end of code download).

NOTE

Serial download is unavailable to the last four words of system RAM

- When using the BAM Serial boot download feature, the BAM initializes an additional four 32-bit words after the end of the downloaded records. This is done to ensure that if the core fetches the last instruction of the downloaded code from the internal SRAM while executing the code, it will not prefetch instructions from memory locations that have not been initialized. If the download image has the exact same size as the internal SRAM, the 20 bytes at the beginning of the SRAM will be written with zero value due to incomplete memory decoding. So, when using the serial download feature of the BAM, make sure that the maximum address of the downloaded code does not exceed the end address of the SRAM minus 16 bytes.

Chapter 10
Interrupts and Interrupt Controller (INTC)

10.1 Introduction

This chapter describes the interrupts and the interrupt controller (INTC), which schedules interrupt requests (IRQs) from software and internal peripherals to the e200z6 and e200z0 cores. The INTC provides interrupt prioritization and preemption, interrupt masking, interrupt priority elevation, and protocol support. The INTC supports 316 interrupt requests.

The INTC has two independent sets of priority arbitration/comparison, request selection, vector encoder and acknowledge logic—one set for each CPU. This allows each CPU to handle its software-assigned interrupt requests independently of the other CPU's operation, and provides flexibility for the user to decide which core should handle which interrupt sources in the application. This flexibility comes from a set of configuration bits that allows any interrupt source to generate an interrupt request to either the Z6 or Z0 or to both the Z6 and Z0 cores.

When multiple tasks share a resource, coherent accesses to that resource need to be supported. The INTC supports the priority ceiling protocol for coherent accesses. By providing a modifiable priority mask, the priority can be raised temporarily so that all tasks which share the resource cannot preempt each other.

Multiple processors can assert interrupt requests to each other through software settable interrupt requests, i.e., by using application software to assert an interrupt request. These same software settable interrupt requests also can be used to break the work involved in servicing an interrupt request into a high priority portion and a low priority portion. The high priority portion is initiated by a peripheral interrupt request, but then the ISR can assert a software settable interrupt request to finish the servicing in a lower priority ISR.

10.1.1 Block Diagram

Interrupts implemented by the MCU are defined in the *e200z6 PowerPC™ Core Reference Manual*.

Figure 10-1 shows a block diagram of the interrupt controller (INTC).

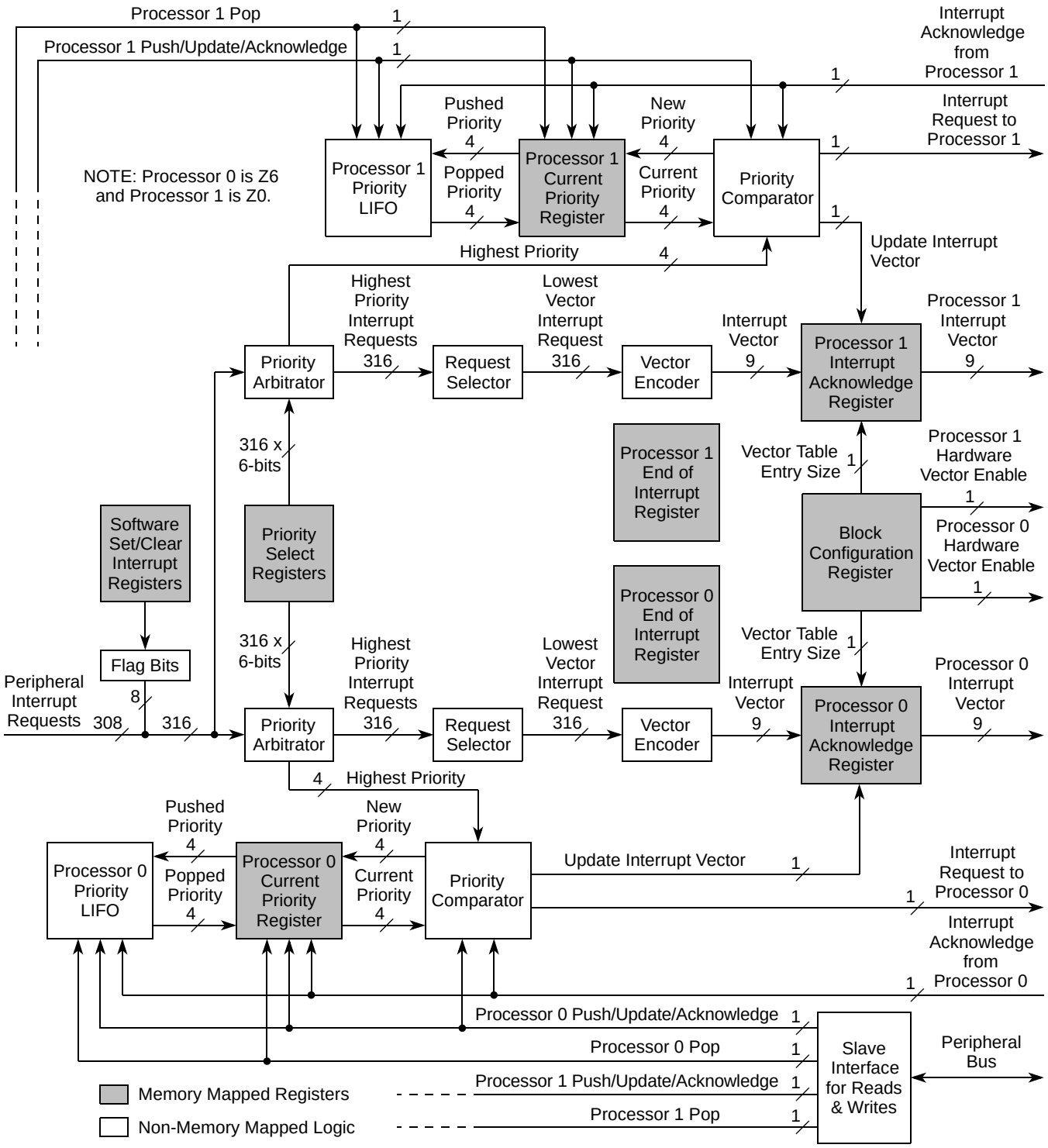


Figure 10-1. INTC Block Diagram

10.1.2 Interrupt Controller Features

- Supports 308 peripheral and eight software-settable interrupt request sources.
- Each interrupt source can be steered by software to processor 0 (Z6), processor 1 (Z0), or both processors interrupt request outputs.

NOTE

By default, processor 0 (Z6) receives all interrupt requests, so backward compatibility with single processor systems is maintained.

- 9-bit unique vector for each interrupt request source in hardware vector mode.
- Each interrupt source can be programmed to one of 16 priorities
- Preemption
  - Preemptive prioritized interrupt requests to processor
  - ISR at a higher priority preempts ISRs or tasks at lower priorities
  - Automatic pushing or popping of preempted priority to or from a LIFO
  - Ability to modify the ISR or task priority; modifying the priority can be used to implement the priority ceiling protocol for accessing shared resources.
- Low latency—three clocks from receipt of interrupt request from peripheral to interrupt request to processor.

10.1.3 Modes of Operation

The interrupt controller has two handshaking modes with the processor: software vector mode and hardware vector mode. The state of the hardware vector enable bit, INTC_MCR[HVEN_PRC*n*], independently determines which mode is used for each CPU.

In debug mode the interrupt controller operation is identical to its normal operation of software vector mode or hardware vector mode.

10.1.3.1 Software Vector Mode

In software vector mode, as shown in Figure 10-2, the CPU branches to a common interrupt exception handler whose location is determined by an address derived from special purpose registers IVPR and IVOR4. The interrupt exception handler reads the INTC_IACKR to determine the vector of the interrupt request source.

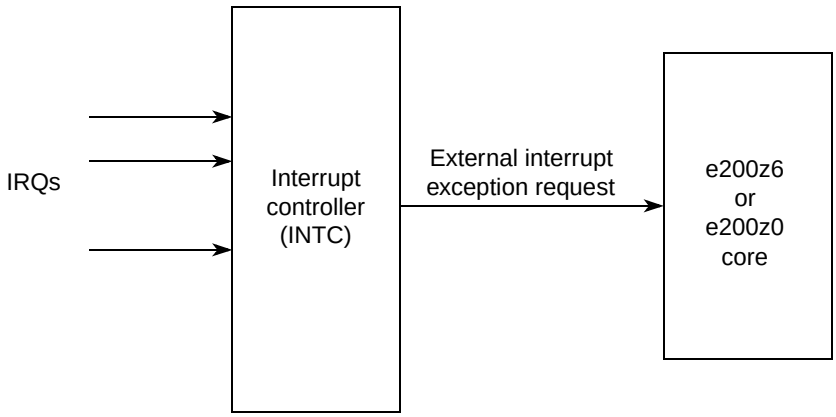
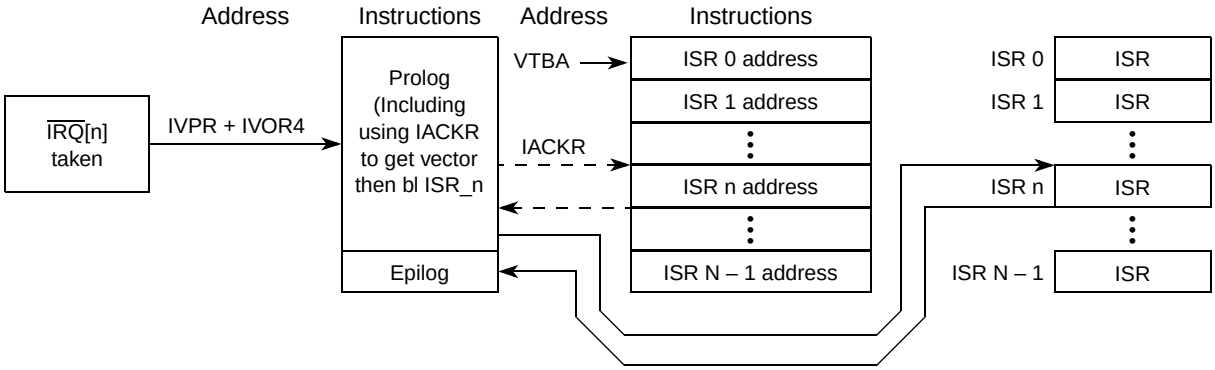


Figure 10-2. INTC Software Vector Mode

Typical program flow for software vector mode is shown in Figure 10-3.



N is the maximum number of usable interrupt vectors, which equals 316, and includes 26 reserved IRQ vectors and eight software-settable IRQ vectors.

Figure 10-3. Program Flow–Software Vector Mode

The common interrupt exception handler address is calculated by hardware as shown in Figure 10-4 for the Z0 core and Figure 10-5 for the Z6 core. The upper half of the interrupt vector prefix register (IVPR) is added to the offset contained in the external input interrupt vector offset register (IVOR4).

NOTE

Since bits IVOR4[28:31] are not part of the offset value for the Z6, the vector offset must be located on a quad-word (16-byte) aligned location in memory. For the Z0 core, the value of IVOR4 is hard coded to 0x040.

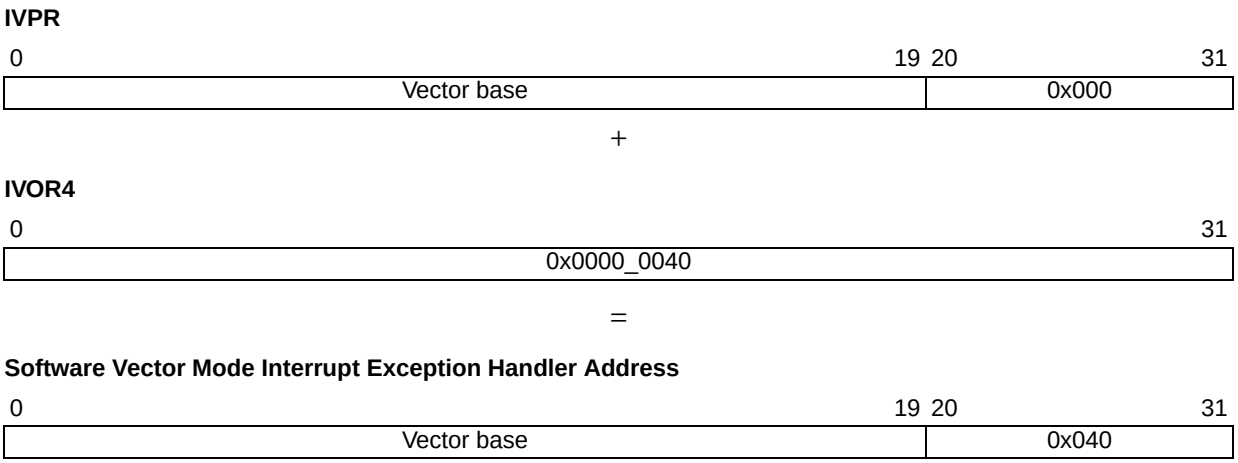


Figure 10-4. Z0 Software Vector Mode: Interrupt Exception Handler Address Calculation

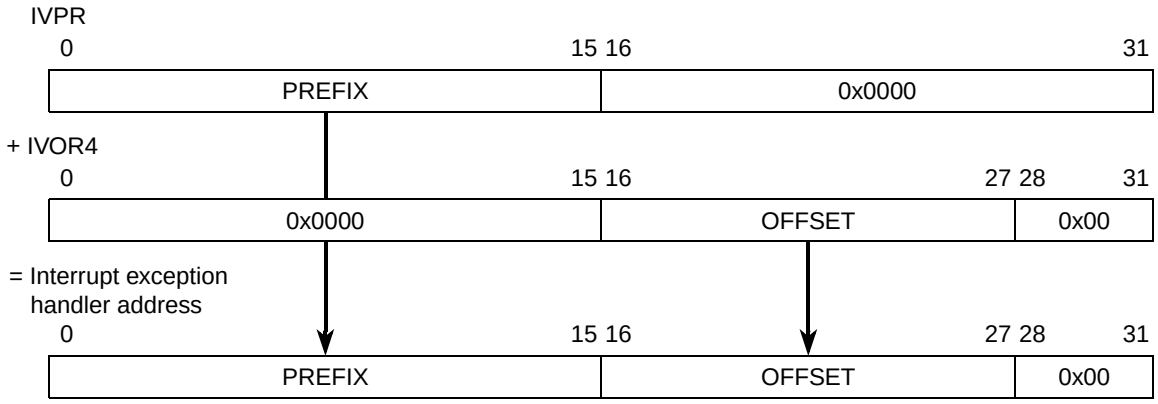


Figure 10-5. Z6 Software Vector Mode: Interrupt Exception Handler Address Calculation

As shown in Figure 10-3, the common interrupt exception handler reads the INTC_IACKR_PRC*n* to determine the vector of the interrupt request source. The INTC_IACKR_PRC*n* register contains a 32-bit address for a vector table base address (VTBA) plus an offset to access the interrupt vector (INTVEC). The address is then used to branch to the corresponding routine for that peripheral or software interrupt source.

Reading the INTC_IACKR_PRC*n* acknowledges the INTC's interrupt request and negates the interrupt request to the processor. The interrupt request to the processor does not clear if a higher priority interrupt request arrives. Even in this case, INTVEC does not update to the higher priority request until the lower priority interrupt request is acknowledged by reading the INTC_IACKR_PRC*n*. The reading also pushes the PRI value in the INTC current priority register (INTC_CPR_PRC*n*) onto the LIFO and updates PRI in the INTC_CPR_PRC*n* with the priority of the interrupt request. The INTC_CPR_PRC*n* masks any peripheral or software settable interrupt request at the same or lower priority of the current value of the PRI field in INTC_CPR_PRC*n* from generating an interrupt request to the processor.

The interrupt exception handler must write to the end-of-interrupt register (INTC_EOIR_PRC*n*) to complete the operation. Writing to the INTC_EOIR_PRC*n* ends the servicing of the interrupt request. The INTC's LIFO is popped into the INTC_CPR_PRC*n*'s PRI field by writing to the INTC_EOIR_PRC*n*, and

the size of a write does not affect the operation of the write. Those values and sizes written to this register neither update the INTC_EOIR_PRC*n* contents nor affect whether the LIFO pops. For possible future compatibility, write four bytes of all 0s to the INTC_EOIR_PRC*n*. The timing relationship between popping the LIFO and disabling recognition of external input has no restriction. The writes can happen in either order.

However, disabling recognition of the external input before popping the LIFO eases the calculation of the maximum stack depth at the cost of postponing the servicing of the next interrupt request.

10.1.3.2 Hardware Vector Mode

For high priority interrupt requests, the time from the assertion of the interrupt request from the peripheral to when the processor is performing useful work to service the interrupt request needs to be minimized. The INTC can be optimized to support this goal through the hardware vector mode, where a unique vector is provided for each interrupt request source. It also provides 16 priorities so that lower priority ISRs do not delay the execution of higher priority ISRs. Since each individual application has different priorities for each source of interrupt request, the priority of each interrupt request is configurable.

Typical program flow for hardware vector mode is shown in Figure 10-6.

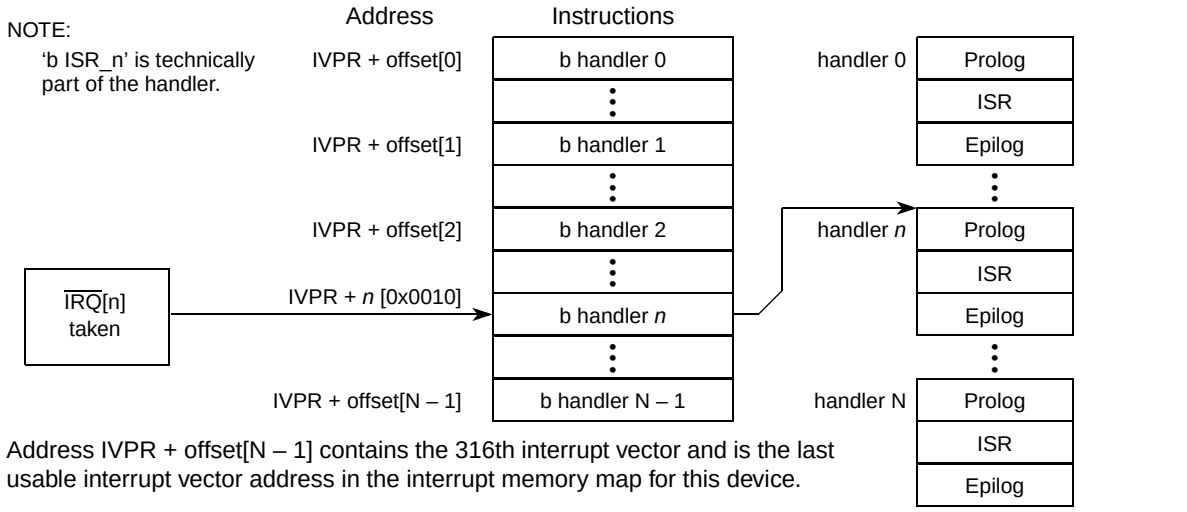


Figure 10-6. Program Flow–Hardware Vector Mode

In hardware vector mode, the interrupt exception handler address is specific to the peripheral or software settable interrupt source rather than being common to all of them. No IVOR is used. The interrupt exception handler address is calculated by hardware as shown in Figure 10-7 for the Z0 core and in Figure 10-8 for the Z6 core. The upper half of the interrupt vector prefix register (IVPR) is added to an offset which corresponds to the peripheral or software interrupt source which caused the interrupt request. The offset matches the value in the Interrupt Vector field, INTC_IACKR_PRC*n*[INTVEC]. Each interrupt exception handler address is aligned on a quad word (16-byte) boundary for the Z6 and on a word boundary (4-byte) for the Z0. IVOR4 is not used in this mode, and software does not need to read INTC_IACKR_PRC*n* to get the interrupt vector number.

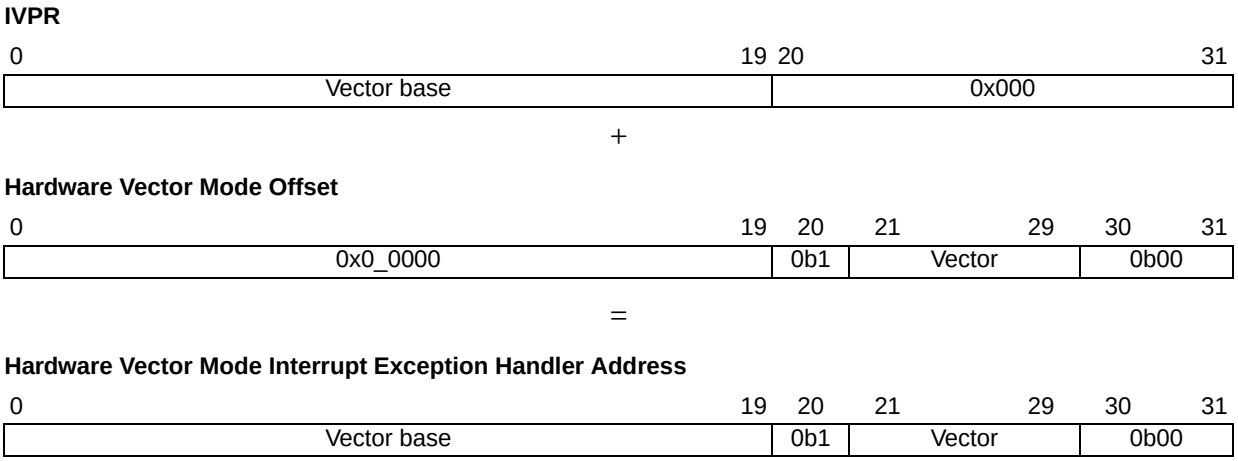


Figure 10-7. Z0 Hardware Vector Mode: Interrupt Exception Handler Address Calculation

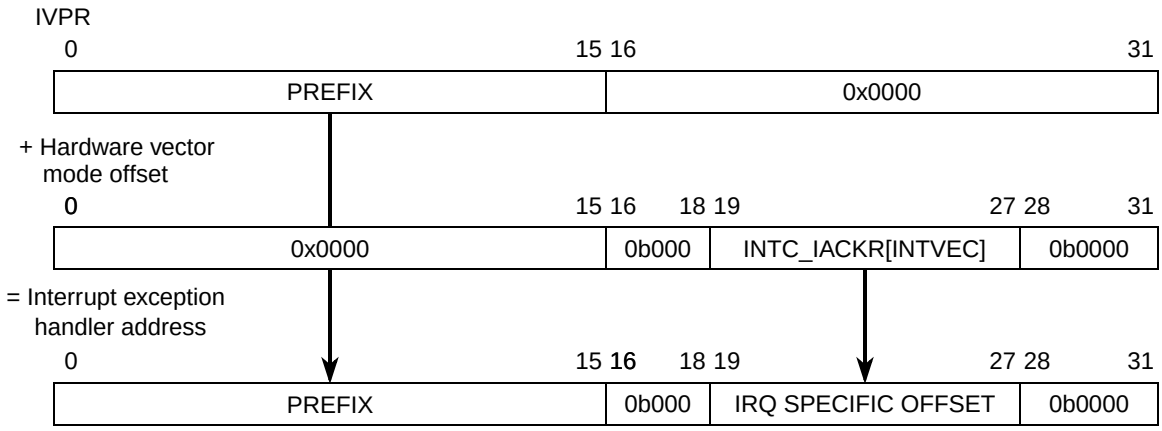


Figure 10-8. Z6 Hardware Vector Mode: Interrupt Exception Handler Address Calculation

The processor negates INTC's interrupt request when automatically acknowledging the interrupt request. However, the interrupt request to the processor do not negate if a higher priority interrupt request arrives. Even in this case, the interrupt vector number does not update to the higher priority request until the lower priority request is acknowledged by the processor.

The assertion of the interrupt acknowledge signal pushes the PRI value in the INTC_CPR_PRC*n* onto the LIFO and updates PRI in the INTC_CPR_PRC*n* with the new priority.

10.2 External Signal Description

The INTC has no direct external MCU signals. However, there are external pins that can be configured in the SIU as external interrupt request input pins. When configured in this function, an interrupt on the pin sets an external interrupt flag. These flags can cause one of five peripheral interrupt requests to the interrupt controller.

For more information on external interrupts, the pins used, and how to configure them, refer to Chapter 3, Signal Description, and Chapter 8, System Integration Unit (SIU), for more information on these pins.

10.3 Memory Map and Registers

10.3.1 INTC Memory Map

Table 10-1 shows the INTC memory map.

Table 10-1. INTC Memory Map

Offset from INTC_BASE_ADDR (0xFFF4_8000)	Register	Access	Reset Value	Section/Page
0x0000	INTC_MCR—INTC module configuration register	R/W	0x0000_0000	10.3.2.1/10-9
0x0004	Reserved			
0x0008	INTC_CPR_PRC0—INTC current priority register for processor 0 (Z6)	R/W	0x0000_000F	10.3.2.2/10-10
0x00C	INTC_CPR_PRC1—INTC current priority register for processor 1 (Z0)	R/W	0x0000_000F	10.3.2.3/10-12
0x0010	INTC_IACKR_PRC0—INTC interrupt acknowledge register for processor 0 (Z6)	R[1]/W	0x0000_0000	10.3.2.4/10-12
0x0014	INTC_IACKR_PRC1—INTC interrupt acknowledge register for processor 1 (Z0)	R[1]/W	0x0000_0000	10.3.2.5/10-14
0x0018	INTC_EOIR_PRC0—INTC end of interrupt register for processor 0 (Z6)	W	0x0000_0000	10.3.2.6/10-14
0x001C	INTC_EOIR_PRC1—INTC end of interrupt register for processor 1 (Z0)	W	0x0000_0000	10.3.2.7/10-15
0x0020	INTC_SSCIR0_3—INTC software set/clear interrupt register 0–3	R/W	0x0000_0000	10.3.2.8/10-15
0x0024	INTC_SSCIR4_7—INTC software set/clear interrupt register 4–7	R/W	0x0000_0000	10.3.2.8/10-15
0x0028 – 0x003F	Reserved			
0x0040 – 0x017B[2]	INTC_PSR0_3—INTC priority select register 0 – 3 to INTC_PSR312_315 — INTC priority select register 312 – 315	R/W	0x0000_0000	10.3.2.9/10-16
0x017C – 0x3FFF	Reserved			

1 When the HVEN bit in the INTC module configuration register (INTC_MCR) is asserted, a read of the INTC_IACKR_PRC*n* has no side effects.

2 A complete list of address offsets for INTC_PSR is provided in Table 10-10 and in Table A-4.

NOTE

To ensure compatibility with all PowerPC processors, the TLB entry covering the INTC memory map must be configured as guarded, both in software and hardware vector modes.

- In software vector mode, the INTC_IACKR must not be read speculatively.

- In hardware vector mode, guarded writes to the INTC_CPR or INTC_EOIR complete before the interrupt acknowledge signal from the processor asserts.

10.3.2 Register Descriptions

With the exception of the INTC_SSCI*n* and INTC_PSR*n* registers, all registers are 32 bits in width. Any combination of accessing the four bytes of a register with a single access is supported, provided that the access does not cross a register boundary. These supported accesses include types and sizes of eight bits, aligned 16 bits, misaligned 16 bits to the middle two bytes, and aligned 32 bits.Although INTC_SSCI*n* and INTC_PSR*n* are 8 bits wide, they can be accessed with a single 16-bit or 32-bit access, provided that the access does not cross a 32-bit boundary.

In software vector mode, the side effects of a read of INTC_IACKR_PRC0 and INTC_IACR_PRC1 are the same regardless of the size of the read. In either software or hardware vector mode, the size of a write to either INTC_EOIR_PRC0 or INTC_EOIR_PRC1 does not affect the operation of the write.

10.3.2.1 INTC Module Configuration Register (INTC_MCR)

The module configuration register is used to configure options of the INTC.

Offset: INTC_BASE_ADDR + 0x0000 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	VTES_PRC1	0	0	0	0	HVEN_PRC1	0	0	VTES_PRC0	0	0	0	0	HVEN_PRC0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 10-9. INTC Module Configuration Register (INTC_MCR)

Table 10-2. INTC_MCR Field Descriptions

Field	Description
VTES_PRC1	For software mode only, the Vector Table Entry Size for Processor 1 (Z0). The VTES_PRC1 bit controls the number of 0s to the right of INTVEC_PRC1 in INTC_IACKR_PRC1. If the contents of INTC_IACKR_PRC1 are used as an address of an entry in a vector table, then the number of rightmost 0s will determine the size of each vector table entry. 0 4 bytes. 1 8 bytes.
HVEN_PRC1	Hardware Vector Enable for Processor 1 (Z0). The HVEN bit controls whether the INTC is in hardware vector mode or software vector mode. Refer to Section 10.1.3, Modes of Operation, for details of handshaking with the processor in each mode. 0 Software vector mode. 1 Hardware vector mode.

Table 10-2. INTC_MCR Field Descriptions (continued)

Field	Description
VTES_PRC0	For software mode only, the Vector Table Entry Size for Processor 0 (Z6). The VTES_PRC0 bit controls the number of 0s to the right of INTVEC_PRC0 in INTC_IACKR_PRC0. If the contents of INTC_IACKR_PRC0 are used as an address of an entry in a vector table, then the number of rightmost 0s will determine the size of each vector table entry. 0 4 bytes. 1 8 bytes.
HVEN_PRC0	Hardware Vector Enable for Processor 0 (Z6). The HVEN bit controls whether the INTC is in hardware vector mode or software vector mode. Refer to Section 10.1.3, Modes of Operation, for details of handshaking with the processor in each mode. 0 Software vector mode. 1 Hardware vector mode.

10.3.2.2 INTC Current Priority Register for Processor 0 (Z6) (INTC_CPR_PRC0)

Offset: INTC_BASE_ADDR + 0x0008 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	PRI			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1

Figure 10-10. INTC Current Priority Register for Processor 0 (Z6) (INTC_CPR_PRC0)

Table 10-3. INTC_CPR_PRC0 Field Descriptions

Field	Description
PRI	Priority. PRI is the priority of the currently executing Z6 ISR according to the field values defined in Table 10-4.

The current priority register masks any peripheral or software settable interrupt request at the same or lower priority of the current value than the PRI field in INTC_CPR_PRC0 from generating an interrupt request to processor 0 (Z6). When INTC_IACKR_PRC0 is read in software vector mode, or the interrupt acknowledge signal from the processor is asserted in hardware vector mode, the value of PRI is pushed onto the LIFO, and PRI is updated with the priority of the preempting interrupt request. When INTC_EOIR_PRC0 is written, the LIFO is popped into the INTC_CPR_PRC0's PRI field. An exception case in hardware vector mode to this behavior is described in Section 10.1.3.2, Hardware Vector Mode.

The masking priority can be raised or lowered by writing to the PRI field, supporting the PCP. Refer to Section 10.5.5, Priority Ceiling Protocol.

NOTE

A store to raise the PRI field which closely precedes an access to a shared resource can result in a non-coherent access to that resource unless an **mbar** or **msync** followed by an **isync** sequence of instructions is executed between the accesses. An **mbar** or **msync** instruction is also necessary after accessing the resource but before lowering the PRI field. Refer to Section 10.5.5.2, Ensuring Coherency, for example code to ensure coherency.

Table 10-4. PRI Values

PRI	Meaning
1111	Priority 15—highest priority
1110	Priority 14
1101	Priority 13
1100	Priority 12
1011	Priority 11
1010	Priority 10
1001	Priority 9
1000	Priority 8
0111	Priority 7
0110	Priority 6
0101	Priority 5
0100	Priority 4
0011	Priority 3
0010	Priority 2
0001	Priority 1
0000	Priority 0—lowest priority

10.3.2.3 INTC Current Priority Register for Processor 1 (Z0) (INTC_CPR_PRC1)

Offset: INTC_BASE_ADDR + 0x000C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	PRI			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1

Figure 10-11. INTC Current Priority Register for Processor 1 (Z0) (INTC_CPR_PRC1)

Table 10-5. INTC_CPR_PRC1 Field Descriptions

Field	Description
PRI	Priority. The function of this register is the same as described for processor 0 (Z6) in Section 10.3.2.2, INTC Current Priority Register for Processor 0 (Z6) (INTC_CPR_PRC0).

10.3.2.4 INTC Interrupt Acknowledge Register for Processor 0 (Z6) (INTC_IACKR_PRC0)

Offset: INTC_BASE_ADDR + 0x0010 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	VTBA_PRC0 (most significant 16 bits)															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	VTBA_PRC0 (least significant five bits)				INTVEC_PRC0[1]										0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

[1] When the VTES_PRC0 bit in INTC_MCR is asserted, INTVEC_PRC0 is shifted to the left one bit. Bit 29 is read as a 0. VTBA_PRC0 is narrowed to 20 bits in width.

Figure 10-12. INTC Interrupt Acknowledge Register for Processor 0 (Z6) (INTC_IACKR_PRC0)

Table 10-6. INTC_IACKR_PRC0 Field Descriptions

Field	Description
VTBA_PRC0	Vector Table Base Address for Processor 0 (Z6). VTBA_PRC0 can be the base address of a vector table of addresses of ISRs for processor 0 (Z6). The VTBA_PRC0 only uses the leftmost 20 bits when the VTES_PRC0 bit in INTC_MCR is asserted.
INTVEC_PRC0	Interrupt Vector for Processor 0 (Z6). INTVEC_PRC0 is the vector of the peripheral or software settable interrupt request that caused the interrupt request to the processor. When the interrupt request to the processor asserts, the INTVEC_PRC0 is updated, whether the INTC is in software or hardware vector mode.

The INTC_IACKR_PRC*n* provides a value that can be used to load the address of an ISR from a vector table. The vector table can be composed of addresses of the ISRs specific to their respective interrupt vectors.

In software vector mode, reading the INTC_IACKR_PRC0 acknowledges the INTC's interrupt request. Refer to Section 10.1.3, Modes of Operation, for a detailed description of the effect on the interrupt request to the processor. The reading also pushes the PRI value in the INTC current priority register (INTC_CPR_PRC*n*) onto the LIFO and updates PRI in the INTC_CPR_PRC*n* with the priority of the interrupt request. The side effect from the reads in software vector mode, that is, the effect on the interrupt request to the processor, the current priority, and the LIFO, are the same regardless of the size of the read

Reading the INTC_IACKR_PRC*n* does not have side effects in hardware vector mode.

NOTE

The INTC_IACKR_PRC*n* must not be read speculatively while in software vector mode. Therefore, for future compatibility, the TLB entry covering the INTC_IACKR_PRC*n* must be configured to be guarded.

In software vector mode, the INTC_IACKR_PRC*n* must be read before setting MSR[EE]. No synchronization instruction is needed after reading the INTC_IACKR_PRC*n* and before setting MSR[EE].

However, the time for the processor to recognize the assertion or negation of the external input to it is not defined by the book E architecture and can be greater than 0. Therefore, insert instructions between the reading of the INTC_IACKR_PRC*n* and the setting of MSR[EE] that consumes at least two processor clock cycles. This length of time allows the interrupt request negation to propagate through the processor before MSR[EE] is set.

10.3.2.5 INTC Interrupt Acknowledge Register for Processor 1 (Z0) (INTC_IACKR_PRC1)

Offset: INTC_BASE_ADDR + 0x0014 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	VTBA_PRC1 (most significant 16 bits)															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	VTBA_PRC1 (5 least-significant bits)				INTVEC_PRC1[1]										0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

1 When the VTES_PRC1 bit in INTC_MCR is asserted, INTVEC_PRC1 is shifted to the left one bit. Bit 29 is read as 0. VTBA_PRC1 is narrowed to 20 bits wide

Figure 10-13. INTC Interrupt Acknowledge Register for Processor 1 (Z0) (INTC_IACKR_PRC1)

Table 10-7. INTC_IACKR_PRC1 Field Descriptions

Field	Description
VTBA_PRC1	Vector Table Base Address for Processor 1 (Z0). VTBA_PRC1 can be the base address of a vector table of addresses of ISRs for processor 1 (Z0). The VTBA_PRC1 only uses the leftmost 20 bits when the VTES_PRC1 bit in INTC_MCR is asserted.
INTVEC_PRC1	Interrupt Vector for Processor 1 (Z0). INTVEC_PRC1 is the vector of the peripheral or software settable interrupt request that caused the interrupt request to the processor. When the interrupt request to the processor asserts, the INTVEC_PRC1 is updated, whether the INTC is in software or hardware vector mode.

10.3.2.6 INTC End-of-Interrupt Register for Processor 0 (Z6) (INTC_EOIR_PRC0)

Offset: INTC_BASE_ADDR + 0x0018 Access: User write-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R																
W	INTC_EOIR_PRC0															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R																
W	INTC_EOIR_PRC0															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 10-14. INTC End-of-Interrupt Register for Processor 0 (Z6) (INTC_EOIR_PRC0)

Writing to the end-of-interrupt register signals the end of the servicing of the interrupt request. When the INTC_EOIR_PRC0 is written, the priority last pushed on the LIFO is popped into INTC_CPR_PRC0. An exception to this behavior is described in Section 10.1.3.2, Hardware Vector Mode. The values and size of data written to the INTC_EOIR_PRC0 are ignored. The values and sizes written to this register neither update the INTC_EOIR_PRC0 contents or affect whether the LIFO pops. For possible future compatibility, write four bytes of all 0s to the INTC_EOIR_PRC0.

10.3.2.7 INTC End-of-Interrupt Register for Processor 1 (Z0) (INTC_EOIR_PRC1)

Offset: INTC_BASE_ADDR + 0x001C Access: User write-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R																
W	INTC_EOIR_PRC1															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R																
W	INTC_EOIR_PRC1															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 10-15. INTC End-of-Interrupt Register for Processor 1 (Z0) (INTC_EOIR_PRC1)

The register's function is the same as for processor 0 (Z6) as described in Section 10.3.2.6, INTC End-of-Interrupt Register for Processor 0 (Z6) (INTC_EOIR_PRC0).

10.3.2.8 INTC Software Set/Clear Interrupt Registers (INTC_SSCIR0_3–INTC_SSCIR4_7)

Offset: INTC_BASE_ADDR + 0x0020 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	CLR0	0	0	0	0	0	0	0	CLR1
W							SET0								SET1	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	CLR2	0	0	0	0	0	0	0	CLR3
W							SET2								SET3	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 10-16. INTC Software Set/Clear Interrupt Register 0–3 (INTC_SSCIR[0:3])

Offset: INTC_BASE_ADDR + 0x0024 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	CLR4	0	0	0	0	0	0	0	CLR5
W							SET4								SET5	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	CLR6	0	0	0	0	0	0	0	CLR7
W							SET6								SET7	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 10-17. INTC Software Set/Clear Interrupt Register 4–7 (INTC_SSCIR[4:7])

Table 10-8. INTC_SSCIR[0:7] Field Descriptions

Field	Description
SET	Set Flag Bits. Writing a 1 sets the corresponding CLR*n* bit. Writing a 0 has no effect. Each SET*n* is always read as a 0.
CLR	Clear Flag Bits. CLR*n* is the flag bit. Writing a 1 to CLR*nx* clears it provided that a 1 is not written simultaneously to its corresponding SET*n* bit. Writing a 0 to CLR*n* has no effect. 0 Interrupt request not pending within INTC. 1 Interrupt request pending within INTC.

The software set/clear interrupt registers support the setting or clearing of software settable interrupt request. These registers contain eight independent sets of bits to set and clear a corresponding flag bit by software. Excepting being set by software, this flag bit behaves the same as a flag bit set within a peripheral. This flag bit generates an interrupt request within the INTC like a peripheral interrupt request. Writing a 1 to SET*n* leaves SET*n* unchanged at 0 but sets CLR*n*. Writing a 0 to SET*n* has no effect. CLR*n* is the flag bit. Writing a 1 to CLR*n* clears it. Writing a 0 to CLR*n* has no effect. If a 1 is written simultaneously to a pair of SET*n* and CLR*n* bits, CLR*n* is asserted, regardless of whether CLR*n* was asserted before the write.

10.3.2.9 INTC Priority Select Registers (INTC_PSR0_3–INTC_PSR312_315)

Offset: INTC_BASE_ADDR + 0x0040 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	PRC_SEL0		0	0	PRI0				PRC_SEL1		0	0	PRI1			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	PRC_SEL2		0	0	PRI2				PRC_SEL3		0	0	PRI3			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 10-18. INTC Priority Select Register 0–3 (INTC_PSR0–3)

Offset: INTC_BASE_ADDR + 0x0178 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	PRC_ SEL312		0	0	PRI312				PRC_ SEL313		0	0	PRI313			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	PRC_ SEL314		0	0	PRI314				PRC_ SEL315		0	0	PRI315			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 10-19. INTC Priority Select Register 312–315 (INTC_PSR312–315)

Table 10-9. INTC_PSR0_3–INTC_PSR312_315 Field Descriptions

Field	Description
PRC_SEL0–PRC_SEL315	Processor Select. If an interrupt source is enabled, PRC_SEL*n* selects whether the interrupt request is to be sent to processor 0 (Z6), processor 1 (Z0), or both. See Table 10-11.
PRI0–PRI315	Priority Select. PRI*n* selects the priority for interrupt requests. Refer to Section 10.4.2, Priority Management.

Table 10-10. INTC Priority Select Register Address Offsets

INTC_PSR*n*_*n*	Offset Address
INTC_PSR0_3	0x0040
INTC_PSR4_7	0x0044
INTC_PSR8_11	0x0048
INTC_PSR12_15	0x004C
INTC_PSR16_19	0x0050
INTC_PSR20_23	0x0054
INTC_PSR24_27	0x0058
INTC_PSR28_31	0x005C
INTC_PSR32_35	0x0060
INTC_PSR36_39	0x0064
INTC_PSR40_43	0x0068
INTC_PSR44_47	0x006C
INTC_PSR48_51	0x0070
INTC_PSR52_55	0x0074
INTC_PSR56_59	0x0078
INTC_PSR60_63	0x007C
INTC_PSR64_67	0x0080
INTC_PSR68_71	0x0084
INTC_PSR72_75	0x0088
INTC_PSR76_79	0x008C
INTC_PSR80_83	0x0090
INTC_PSR84_87	0x0094
INTC_PSR88_91	0x0098
INTC_PSR92_95	0x009C
INTC_PSR96_99	0x00A0
INTC_PSR100_103	0x00A4
INTC_PSR104_107	0x00A8

INTC_PSR*n*_*n*	Offset Address
INTC_PSR160_163	0x00E0
INTC_PSR164_167	0x00E4
INTC_PSR168_171	0x00E8
INTC_PSR172_175	0x00EC
INTC_PSR176_179	0x00F0
INTC_PSR180_183	0x00F4
INTC_PSR184_187	0x00F8
INTC_PSR188_191	0x00FC
INTC_PSR192_195	0x0100
INTC_PSR196_199	0x0104
INTC_PSR200_203	0x0108
INTC_PSR204_207	0x010C
INTC_PSR208_211	0x0110
INTC_PSR212_215	0x0114
INTC_PSR216_219	0x0118
INTC_PSR220_223	0x011C
INTC_PSR224_227	0x0120
INTC_PSR228_231	0x0124
INTC_PSR232_235	0x0128
INTC_PSR236_239	0x012C
INTC_PSR240_243	0x0130
INTC_PSR244_247	0x0134
INTC_PSR248_251	0x0138
INTC_PSR252_255	0x013C
INTC_PSR256_259	0x0140
INTC_PSR260_263	0x0144
INTC_PSR264_267	0x0148

Table 10-10. INTC Priority Select Register Address Offsets

INTC_PSR*n*_*n*	Offset Address
INTC_PSR108_111	0x00AC
INTC_PSR112_115	0x00B0
INTC_PSR116_119	0x00B4
INTC_PSR120_123	0x00B8
INTC_PSR124_127	0x00BC
INTC_PSR128_131	0x00C0
INTC_PSR132_135	0x00C4
INTC_PSR136_139	0x00C8
INTC_PSR140_143	0x00CC
INTC_PSR144_147	0x00D0
INTC_PSR148_151	0x00D4
INTC_PSR152_155	0x00D8
INTC_PSR156_159	0x00DC

INTC_PSR*n*_*n*	Offset Address
INTC_PSR268_271	0x014C
INTC_PSR272_275	0x0150
INTC_PSR276_279	0x0154
INTC_PSR280_283	0x0158
INTC_PSR284_287	0x015C
INTC_PSR288_291	0x0160
INTC_PSR292_295	0x0164
INTC_PSR296_299	0x0168
INTC_PSR300_303	0x016C
INTC_PSR304_307	0x0170
INTC_PSR308_311	0x0174
INTC_PSR312_315	0x0178

The priority select registers support the selection of an individual priority for each source of interrupt request, and whether the interrupt request is to be sent to processor 0 (Z6), processor 1, (Z0) or both. The unique vector of each peripheral or software settable interrupt request determines which INTC_PSR*n*_*n* is assigned to that interrupt request. The software settable interrupt requests 0–7 are assigned vectors 0–7, and their priorities are configured in INTC_PSR0_3 and INTC_PSR4_7, respectively. The peripheral interrupt requests are assigned vectors 8–315, and their priorities are configured in INTC_PSR8_11 through INTC_PSR312_315, respectively (see Section 10.4.1, External Interrupt Request Sources,).

NOTE

The PRC_SEL*n* or PRI*n* field of an INTC_PSR*n*_*n* must not be modified while the corresponding peripheral or software settable interrupt request is asserted.

Table 10-11. Selected Processor for Interrupt Request

PRC_SEL*n*	Meaning
00	Interrupt request sent to processor 0 (Z6)
01	Interrupt request sent to both processors
10	Reserved
11	Interrupt request sent to processor 1 (Z0)

10.4 Functional Description

10.4.1 External Interrupt Request Sources

The INTC has two types of interrupt requests, peripheral and software settable. The assignments between the interrupt requests from the modules to the vectors for input to the CPU are shown in

Table 10-12. The e200z6 and e200z0 Hardware Vector Mode Offset columns list the IRQ-specific offsets when using hardware vector mode

It is important to note that interrupt table entries are 16 bytes in size for the e200z6. For the e200z0, interrupt table entries are only 4 bytes in size. This difference in size, combined with the different starting offset for the e200z0 interrupt requests, makes it impossible for the two cores to share a single interrupt table. To solve this, either two different interrupt tables need to be created, or software vector mode should be used for external interrupt requests.

Table 10-12. PXN20 External Interrupt Requests

Vector Number	e200z6 Hardware Vector Mode Offset	e200z0 Hardware Vector Mode Offset	PXN20 Vector	Vector Type
0	0x0000	0x0800	Software Interrupt 0	Software
1	0x0010	0x0804	Software Interrupt 1	
2	0x0020	0x0808	Software Interrupt 2	
3	0x0030	0x080C	Software Interrupt 3	
4	0x0040	0x0810	Software Interrupt 4	
5	0x0050	0x0814	Software Interrupt 5	
6	0x0060	0x0818	Software Interrupt 6	
7	0x0070	0x081C	Software Interrupt 7	
8	0x0080	0x0820	SWT Timeout SWT	SWT
9	0x0090	0x0824	ECC Error ECC	ECC

Table 10-12. PXN20 External Interrupt Requests (continued)

Vector Number	e200z6 Hardware Vector Mode Offset	e200z0 Hardware Vector Mode Offset	PXN20 Vector	Vector Type
10	0x00A0	0x0828	eDMA–Error 1	eDMA
11	0x00B0	0x082C	eDMA–0	
12	0x00C0	0x0830	eDMA–1	
13	0x00D0	0x0834	eDMA–2	
14	0x00E0	0x0838	eDMA–3	
15	0x00F0	0x083C	eDMA–4	
16	0x0100	0x0840	eDMA–5	
17	0x0110	0x0844	eDMA–6	
18	0x0120	0x0848	eDMA–7	
19	0x0130	0x084C	eDMA–8	
20	0x0140	0x0850	eDMA–9	
21	0x0150	0x0854	eDMA–10	

Table 10-12. PXN20 External Interrupt Requests (continued)

Vector Number	e200z6 Hardware Vector Mode Offset	e200z0 Hardware Vector Mode Offset	PXN20 Vector	Vector Type
22	0x0160	0x0858	eDMA–11	eDMA (continued)
23	0x0170	0x085C	eDMA–12	
24	0x0180	0x0860	eDMA–13	
25	0x0190	0x0864	eDMA–14	
26	0x01A0	0x0868	eDMA–15	
27	0x01B0	0x086C	eDMA–16	
28	0x01C0	0x0870	eDMA–17	
29	0x01D0	0x0874	eDMA–18	
30	0x01E0	0x0878	eDMA–19	
31	0x01F0	0x087C	eDMA–20	
32	0x0200	0x0880	eDMA–21	
33	0x0210	0x0884	eDMA–22	
34	0x0220	0x0888	eDMA–23	
35	0x0230	0x088C	eDMA–24	
36	0x0240	0x0890	eDMA–25	
37	0x0250	0x0894	eDMA–26	
38	0x0260	0x0898	eDMA–27	
39	0x0270	0x089C	eDMA–28	
40	0x0280	0x08A0	eDMA–29	
41	0x0290	0x08A4	eDMA–30	
42	0x02A0	0x08A8	eDMA–31	
43	0x02B0	0x08AC	Semaphore Interrupt 0	Semaphore
44	0x02C0	0x08B0	Semaphore Interrupt 1	
45	0x02D0	0x08B4	Pin Wakeup Interrupt	CRP
46	0x02E0	0x08B8	API/RTC Interrupt	
47	0x02F0	0x08BC	LVI Interrupt	

Table 10-12. PXN20 External Interrupt Requests (continued)

Vector Number	e200z6 Hardware Vector Mode Offset	e200z0 Hardware Vector Mode Offset	PXN20 Vector	Vector Type
48	0x0300	0x08C0	I^2C_A	I2C (see Interrupt vectors 314–315)
49	0x0310	0x08C4	I^2C_B	
50	0x0320	0x08C8	PLL Loss Of Clock	PLL
51	0x0330	0x08CC	PLL Loss Of Lock	
52	0x0340	0x08D0	SIU Overrun	SIU
53	0x0350	0x08D4	SIU External Interrupt 0	
54	0x0360	0x08D8	SIU External Interrupt 1	
55	0x0370	0x08DC	SIU External Interrupt 2	
56	0x0380	0x08E0	SIU External Interrupt 3	
57	0x0390	0x08E4	SIU External Interrupts 15:4	
58	0x03A0	0x08E8	EMIOS 0	EMIOS channels 0 to 23 (see Interrupt vectors 262–269)
59	0x03B0	0x08EC	EMIOS 1	
60	0x03C0	0x08F0	EMIOS 2	
61	0x03D0	0x08F4	EMIOS 3	
62	0x03E0	0x08F8	EMIOS 4	
63	0x03F0	0x08FC	EMIOS 5	
64	0x0400	0x0900	EMIOS 6	
65	0x0410	0x0904	EMIOS 7	
66	0x0420	0x0908	EMIOS 8	
67	0x0430	0x090C	EMIOS 9	
68	0x0440	0x0910	EMIOS 10	
69	0x0450	0x0914	EMIOS 11	
70	0x0460	0x0918	EMIOS 12	
71	0x0470	0x091C	EMIOS 13	
72	0x0480	0x0920	EMIOS 14	
73	0x0490	0x0924	EMIOS 15	

Table 10-12. PXN20 External Interrupt Requests (continued)

Vector Number	e200z6 Hardware Vector Mode Offset	e200z0 Hardware Vector Mode Offset	PXN20 Vector	Vector Type
74	0x04A0	0x0928	EMIOS 16	EMIOS channels 0 to 23 (see Interrupt vectors 262–269) (continued)
75	0x04B0	0x092C	EMIOS 17	
76	0x04C0	0x0930	EMIOS 18	
77	0x04D0	0x0934	EMIOS 19	
78	0x04E0	0x0938	EMIOS 20	
79	0x04F0	0x093C	EMIOS 21	
80	0x0500	0x0940	EMIOS 22	
81	0x0510	0x0944	EMIOS 23	
82	0x0520	0x0948	ADC_EOC	ADC
83	0x0530	0x094C	ADC_ERR	
84	0x0540	0x0950	ADC_WD	
85	0x0550	0x0954	Reserved ADC_B	
86	0x0560	0x0958	Reserved ADC_B	
87	0x0570	0x095C	Reserved ADC_B	
88	0x0580	0x0960	Reserved ADC_C	
89	0x0590	0x0964	Reserved ADC_C	
90	0x05A0	0x0968	Reserved ADC_C	
91	0x05B0	0x096C	Reserved	Reserved
92	0x05C0	0x0970	CRP Flash Ready	
93	0x05D0	0x0974	CTU Trigger	
94	0x05E0	0x0978	Reserved	
95	0x05F0	0x097C	MLB Channel Interrupts 0–15 combined	MLB_DIM
96	0x0600	0x0980	MLB System Interrupt	
97	0x0610	0x0984	MLB Channel Interrupt 0	
98	0x0620	0x0988	MLB Channel Interrupt 1	
99	0x0630	0x098C	MLB Channel Interrupt 2	
100	0x0640	0x0990	MLB Channel Interrupt 3	

Table 10-12. PXN20 External Interrupt Requests (continued)

Vector Number	e200z6 Hardware Vector Mode Offset	e200z0 Hardware Vector Mode Offset	PXN20 Vector	Vector Type
101	0x0650	0x0994	MLB Channel Interrupt 4	MLB_DIM (continued)
102	0x0660	0x0998	MLB Channel Interrupt 5	
103	0x0670	0x099C	MLB Channel Interrupt 6	
104	0x0680	0x09A0	MLB Channel Interrupt 7	
105	0x0690	0x09A4	MLB Channel Interrupt 8	
106	0x06A0	0x09A8	MLB Channel Interrupt 9	
107	0x06B0	0x09AC	MLB Channel Interrupt 10	
108	0x06C0	0x09B0	MLB Channel Interrupt 11	
109	0x06D0	0x09B4	MLB Channel Interrupt 12	
110	0x06E0	0x09B8	MLB Channel Interrupt 13	
111	0x06F0	0x09BC	MLB Channel Interrupt 14	
112	0x0700	0x09C0	MLB Channel Interrupt 15	
113	0x0710	0x09C4	SCI_A	SCI_A to SCI_D (see Interrupt vectors 270–273)
114	0x0720	0x09C8	SCI_B	
115	0x0730	0x09CC	SCI_C	
116	0x0740	0x09D0	SCI_D	
117	0x0750	0x09D4	DSPI_A FIFO Overflow/Underflow	DSPI_A (see Interrupt vectors 274–283 and 306–313)
118	0x0760	0x09D8	DSPI_A End Of Queue	
119	0x0770	0x09DC	DSPI_A Tx FIFO Fill Request	
120	0x0780	0x09E0	DSPI_A Transfer complete	
121	0x0790	0x09E4	DSPI_A Rx FIFO Drain Request	
122	0x07A0	0x09E8	DSPI_B FIFO Overflow/Underflow	DSPI_B (see Interrupt vectors 274–283 and 306–313)
123	0x07B0	0x09EC	DSPI_B End Of Queue	
124	0x07C0	0x09F0	DSPI_B Tx FIFO Fill Request	
125	0x07D0	0x09F4	DSPI_B Transfer complete	
126	0x07E0	0x09F8	DSPI_B Rx FIFO Drain Request	

Table 10-12. PXN20 External Interrupt Requests (continued)

Vector Number	e200z6 Hardware Vector Mode Offset	e200z0 Hardware Vector Mode Offset	PXN20 Vector	Vector Type
127	0x07F0	0x09FC	FLEXCAN_A Bus Off and Warning	FlexCAN_A
128	0x0800	0x0A00	FLEXCAN_A Error	
129	0x0810	0x0A04	Reserved for FLEXCAN_A Wake Up	
130	0x0820	0x0A08	FLEXCAN_A Buffer 0	
131	0x0830	0x0A0C	FLEXCAN_A Buffer 1	
132	0x0840	0x0A10	FLEXCAN_A Buffer 2	
133	0x0850	0x0A14	FLEXCAN_A Buffer 3	
134	0x0860	0x0A18	FLEXCAN_A Buffer 4	
135	0x0870	0x0A1C	FLEXCAN_A Buffer 5	
136	0x0880	0x0A20	FLEXCAN_A Buffer 6	
137	0x0890	0x0A24	FLEXCAN_A Buffer 7	
138	0x08A0	0x0A28	FLEXCAN_A Buffer 8	
139	0x08B0	0x0A2C	FLEXCAN_A Buffer 9	
140	0x08C0	0x0A30	FLEXCAN_A Buffer 10	
141	0x08D0	0x0A34	FLEXCAN_A Buffer 11	
142	0x08E0	0x0A38	FLEXCAN_A Buffer 12	
143	0x08F0	0x0A3C	FLEXCAN_A Buffer 13	
144	0x0900	0x0A40	FLEXCAN_A Buffer 14	
145	0x0910	0x0A44	FLEXCAN_A Buffer 15	
146	0x0920	0x0A48	FLEXCAN_A Buffer 16-31	
147	0x0930	0x0A4C	FLEXCAN_A Buffer 32-63	
148	0x0940	0x0A50	Reserved–RTI not implemented	Reserved
149	0x0950	0x0A54	PIT1	PIT
150	0x0960	0x0A58	PIT2	
151	0x0970	0x0A5C	PIT3	
152	0x0980	0x0A60	PIT4	
153	0x0990	0x0A64	PIT5	

Table 10-12. PXN20 External Interrupt Requests (continued)

Vector Number	e200z6 Hardware Vector Mode Offset	e200z0 Hardware Vector Mode Offset	PXN20 Vector	Vector Type
154	0x09A0	0x0A68	PIT6	PIT (continued)
155	0x09B0	0x0A6C	PIT7	
156	0x09C0	0x0A70	PIT8	
157	0x09D0	0x0A74	FLEXCAN_B Bus Off and Warning	FlexCAN_B
158	0x09E0	0x0A78	FLEXCAN_B Error	
159	0x09F0	0x0A7C	Reserved for FLEXCAN_B Wake Up	
160	0x0A00	0x0A80	FLEXCAN_B Buffer 0	
161	0x0A10	0x0A84	FLEXCAN_B Buffer 1	
162	0x0A20	0x0A88	FLEXCAN_B Buffer 2	
163	0x0A30	0x0A8C	FLEXCAN_B Buffer 3	
164	0x0A40	0x0A90	FLEXCAN_B Buffer 4	
165	0x0A50	0x0A94	FLEXCAN_B Buffer 5	
166	0x0A60	0x0A98	FLEXCAN_B Buffer 6	
167	0x0A70	0x0A9C	FLEXCAN_B Buffer 7	
168	0x0A80	0x0AA0	FLEXCAN_B Buffer 8	
169	0x0A90	0x0AA4	FLEXCAN_B Buffer 9	
170	0x0AA0	0x0AA8	FLEXCAN_B Buffer 10	
171	0x0AB0	0x0AAC	FLEXCAN_B Buffer 11	
172	0x0AC0	0x0AB0	FLEXCAN_B Buffer 12	
173	0x0AD0	0x0AB4	FLEXCAN_B Buffer 13	
174	0x0AE0	0x0AB8	FLEXCAN_B Buffer 14	
175	0x0AF0	0x0ABC	FLEXCAN_B Buffer 15	
176	0x0B00	0x0AC0	FLEXCAN_B Buffer 16–31	
177	0x0B10	0x0AC4	FLEXCAN_B Buffer 32–63	

Table 10-12. PXN20 External Interrupt Requests (continued)

Vector Number	e200z6 Hardware Vector Mode Offset	e200z0 Hardware Vector Mode Offset	PXN20 Vector	Vector Type
178	0x0B20	0x0AC8	FLEXCAN_C Bus Off and Warning	FlexCAN_C
179	0x0B30	0x0ACC	FLEXCAN_C Error	
180	0x0B40	0x0AD0	Reserved for FLEXCAN_C Wake Up	
181	0x0B50	0x0AD4	FLEXCAN_C Buffer 0	FlexCAN_C (continued)
182	0x0B60	0x0AD8	FLEXCAN_C Buffer 1	
183	0x0B70	0x0ADC	FLEXCAN_C Buffer 2	
184	0x0B80	0x0AE0	FLEXCAN_C Buffer 3	
185	0x0B90	0x0AE4	FLEXCAN_C Buffer 4	
186	0x0BA0	0x0AE8	FLEXCAN_C Buffer 5	
187	0x0BB0	0x0AEC	FLEXCAN_C Buffer 6	
188	0x0BC0	0x0AF0	FLEXCAN_C Buffer 7	
189	0x0BD0	0x0AF4	FLEXCAN_C Buffer 8	
190	0x0BE0	0x0AF8	FLEXCAN_C Buffer 9	
191	0x0BF0	0x0AFC	FLEXCAN_C Buffer 10	
192	0x0C00	0x0B00	FLEXCAN_C Buffer 11	
193	0x0C10	0x0B04	FLEXCAN_C Buffer 12	
194	0x0C20	0x0B08	FLEXCAN_C Buffer 13	
195	0x0C30	0x0B0C	FLEXCAN_C Buffer 14	
196	0x0C40	0x0B10	FLEXCAN_C Buffer 15	
197	0x0C50	0x0B14	FLEXCAN_C Buffer 16–31	
198	0x0C60	0x0B18	FLEXCAN_C Buffer 32–63	

Table 10-12. PXN20 External Interrupt Requests (continued)

Vector Number	e200z6 Hardware Vector Mode Offset	e200z0 Hardware Vector Mode Offset	PXN20 Vector	Vector Type
199	0x0C70	0x0B1C	FLEXCAN_D Bus Off and Warning	FlexCAN_D
200	0x0C80	0x0B20	FLEXCAN_D Error	
201	0x0C90	0x0B24	Reserved for FLEXCAN_D Wake Up	
202	0x0CA0	0x0B28	FLEXCAN_D Buffer 0	
203	0x0CB0	0x0B2C	FLEXCAN_D Buffer 1	
204	0x0CC0	0x0B30	FLEXCAN_D Buffer 2	
205	0x0CD0	0x0B34	FLEXCAN_D Buffer 3	
206	0x0CE0	0x0B38	FLEXCAN_D Buffer 4	
207	0x0CF0	0x0B3C	FLEXCAN_D Buffer 5	
208	0x0D00	0x0B40	FLEXCAN_D Buffer 6	FlexCAN_D (continued)
209	0x0D10	0x0B44	FLEXCAN_D Buffer 7	
210	0x0D20	0x0B48	FLEXCAN_D Buffer 8	
211	0x0D30	0x0B4C	FLEXCAN_D Buffer 9	
212	0x0D40	0x0B50	FLEXCAN_D Buffer 10	
213	0x0D50	0x0B54	FLEXCAN_D Buffer 11	
214	0x0D60	0x0B58	FLEXCAN_D Buffer 12	
215	0x0D70	0x0B5C	FLEXCAN_D Buffer 13	
216	0x0D80	0x0B60	FLEXCAN_D Buffer 14	
217	0x0D90	0x0B64	FLEXCAN_D Buffer 15	
218	0x0DA0	0x0B68	FLEXCAN_D Buffer 16–31	
219	0x0DB0	0x0B6C	FLEXCAN_D Buffer 32–63	

Table 10-12. PXN20 External Interrupt Requests (continued)

Vector Number	e200z6 Hardware Vector Mode Offset	e200z0 Hardware Vector Mode Offset	PXN20 Vector	Vector Type
220	0x0DC0	0x0B70	FLEXCAN_E Bus Off and Warning	FlexCAN_E
221	0x0DD0	0x0B74	FLEXCAN_E Error	
222	0x0DE0	0x0B78	Reserved for FLEXCAN_E Wake Up	
223	0x0DF0	0x0B7C	FLEXCAN_E Buffer 0	
224	0x0E00	0x0B80	FLEXCAN_E Buffer 1	
225	0x0E10	0x0B84	FLEXCAN_E Buffer 2	
226	0x0E20	0x0B88	FLEXCAN_E Buffer 3	
227	0x0E30	0x0B8C	FLEXCAN_E Buffer 4	
228	0x0E40	0x0B90	FLEXCAN_E Buffer 5	
229	0x0E50	0x0B94	FLEXCAN_E Buffer 6	
230	0x0E60	0x0B98	FLEXCAN_E Buffer 7	
231	0x0E70	0x0B9C	FLEXCAN_E Buffer 8	
232	0x0E80	0x0BA0	FLEXCAN_E Buffer 9	
233	0x0E90	0x0BA4	FLEXCAN_E Buffer 10	
234	0x0EA0	0x0BA8	FLEXCAN_E Buffer 11	
235	0x0EB0	0x0BAC	FLEXCAN_E Buffer 12	FlexCAN_E (continued)
236	0x0EC0	0x0BB0	FLEXCAN_E Buffer 13	
237	0x0ED0	0x0BB4	FLEXCAN_E Buffer 14	
238	0x0EE0	0x0BB8	FLEXCAN_E Buffer 15	
239	0x0EF0	0x0BBC	FLEXCAN_E Buffer 16–31	
240	0x0F00	0x0BC0	FLEXCAN_E Buffer 32–63	

Table 10-12. PXN20 External Interrupt Requests (continued)

Vector Number	e200z6 Hardware Vector Mode Offset	e200z0 Hardware Vector Mode Offset	PXN20 Vector	Vector Type
241	0x0F10	0x0BC4	FLEXCAN_F Bus Off and Warning	FlexCAN_F
242	0x0F20	0x0BC8	FLEXCAN_F Error	
243	0x0F30	0x0BCC	Reserved for FLEXCAN_F Wake Up	
244	0x0F40	0x0BD0	FLEXCAN_F Buffer 0	
245	0x0F50	0x0BD4	FLEXCAN_F Buffer 1	
246	0x0F60	0x0BD8	FLEXCAN_F Buffer 2	
247	0x0F70	0x0BDC	FLEXCAN_F Buffer 3	
248	0x0F80	0x0BE0	FLEXCAN_F Buffer 4	
249	0x0F90	0x0BE4	FLEXCAN_F Buffer 5	
250	0x0FA0	0x0BE8	FLEXCAN_F Buffer 6	
251	0x0FB0	0x0BEC	FLEXCAN_F Buffer 7	
252	0x0FC0	0x0BF0	FLEXCAN_F Buffer 8	
253	0x0FD0	0x0BF4	FLEXCAN_F Buffer 9	
254	0x0FE0	0x0BF8	FLEXCAN_F Buffer 10	
255	0x0FF0	0x0BFC	FLEXCAN_F Buffer 11	
256	0x1000	0x0C00	FLEXCAN_F Buffer 12	
257	0x1010	0x0C04	FLEXCAN_F Buffer 13	
258	0x1020	0x0C08	FLEXCAN_F Buffer 14	
259	0x1030	0x0C0C	FLEXCAN_F Buffer 15	
260	0x1040	0x0C10	FLEXCAN_F Buffer 16–31	
261	0x1050	0x0C14	FLEXCAN_F Buffer 32–63	

Table 10-12. PXN20 External Interrupt Requests (continued)

Vector Number	e200z6 Hardware Vector Mode Offset	e200z0 Hardware Vector Mode Offset	PXN20 Vector	Vector Type
262	0x1060	0x0C18	EMIOS_24	EMIOS channels 24 to 31 (see Interrupt vectors 58–81)
263	0x1070	0x0C1C	EMIOS_25	
264	0x1080	0x0C20	EMIOS_26	
265	0x1090	0x0C24	EMIOS_27	
266	0x10A0	0x0C28	EMIOS_28	
267	0x10B0	0x0C2C	EMIOS_29	
268	0x10C0	0x0C30	EMIOS_30	
269	0x10D0	0x0C34	EMIOS_31	
270	0x10E0	0x0C38	SCI_E	SCI_E to SCI_H (see Interrupt vectors 113–116 and 306–313)
271	0x10F0	0x0C3C	SCI_F	
272	0x1100	0x0C40	SCI_G	
273	0x1110	0x0C44	SCI_H	
274	0x1120	0x0C48	DSPI_C FIFO Overflow/Underflow	DSPI_C (see Interrupt vectors 117–126)
275	0x1130	0x0C4C	DSPI_C End Of Queue	
276	0x1140	0x0C50	DSPI_C Tx FIFO Fill Request	
277	0x1150	0x0C54	DSPI_C Transfer complete	
278	0x1160	0x0C58	DSPI_C Rx FIFO Drain Request	
279	0x1170	0x0C5C	DSPI_D FIFO Overflow/Underflow	DSPI_D (see Interrupt vectors 117–126)
280	0x1180	0x0C60	DSPI_D End Of Queue	
281	0x1190	0x0C64	DSPI_D Tx FIFO Fill Request	
282	0x11A0	0x0C68	DSPI_D Transfer complete	
283	0x11B0	0x0C6C	DSPI_D Rx FIFO Drain Request	
284	0x11C0	0x0C70	FlexRay_MIF	FlexRay
285	0x11D0	0x0C74	FlexRay_PRIF	
286	0x11E0	0x0C78	FlexRay_CHIF	
287	0x11F0	0x0C7C	FlexRay_WUP_IF	
288	0x1200	0x0C80	FlexRay_FBNE_F	

Table 10-12. PXN20 External Interrupt Requests (continued)

Vector Number	e200z6 Hardware Vector Mode Offset	e200z0 Hardware Vector Mode Offset	PXN20 Vector	Vector Type
289	0x1210	0x0C84	FlexRay_FANE_F	FlexRay (continued)
290	0x1220	0x0C88	FlexRay_RBIF	
291	0x1230	0x0C8C	FlexRay_TBIF	
292	0x1240	0x0C90	Reserved	Soft MediaLB
293	0x1250	0x0C94	Reserved	
294	0x1260	0x0C98	Match on Channel 0	STM
295	0x1270	0x0C9C	Match on Channel 1	
296	0x1280	0x0CA0	Match on Channel 2	
297	0x1290	0x0CA4	Match on Channel 3	
298	0x12A0	0x0CA8	FEC Transmit	FEC
299	0x12B0	0x0CAC	FEC Receive	
300	0x12C0	0x0CB0	FEC Everything else	
301	0x12D0	0x0CB4	Reserved	Reserved for On Platform
302	0x12E0	0x0CB8	Reserved	
303	0x12F0	0x0CBC	Reserved	
304	0x1300	0x0CC0	Reserved	
305	0x1310	0x0CC4	Reserved	
306	0x1320	0x0CC8	SCI_J	SCI_J to SCI_M (see Interrupt vectors 113–116 and 270–273)
307	0x1330	0x0CCC	SCI_K	
308	0x1340	0x0CD0	SCI_L	
309	0x1350	0x0CD4	SCI_M	
310	0x1360	0x0CD8	Reserved for SCI_N	SCI_N and SCI_P to SCI_R (see Interrupt vectors 113–116 and 270–273)
311	0x1370	0x0CDC	Reserved for SCI_P	
312	0x1380	0x0CE0	Reserved for SCI_Q	
313	0x1390	0x0CE4	Reserved for SCI_R	
314	0x13A0	0x0CE8	I2C_C	I2C (see Interrupt vectors 48–49)
315	0x13B0	0x0CEC	I2C_D	

NOTE

The peripheral or software settable interrupt request asserts when the PRI*n* value in the interrupt priority select register (INTC_PSR*n*) is greater than the PRI*n* value in interrupt current priority register (INTC_CPR).

If an asserted peripheral or software settable interrupt request negates before the processor acknowledges its request, the interrupt request can reassert and remain asserted. If this occurs, the processor uses the INTC_PSR*n* value to locate the IRQ vector, and updates the PRI*n* value in the INTC_CPR with the PRI*n* value in INTC_PSR*n*.

Clearing the peripheral interrupt request enable bit for the peripheral initiating the request, or setting the IRQ mask bit has the same consequences as clearing its flag bit. Setting its enable bit or clearing its mask bit while its flag bit is asserted has the same effect on the INTC as an interrupt event setting the flag bit.

10.4.1.1 Peripheral Interrupt Requests

An interrupt event in a peripheral's hardware sets a flag bit that resides in the peripheral. The interrupt request from the peripheral is driven by that flag bit.

The time from when the peripheral starts to drive its peripheral interrupt request to the INTC to the time that the INTC starts to drive the interrupt request to the processor is three clocks.

Interrupt requests from devices external to the PXN20 are classified as peripheral interrupt requests in this reference manual. These type of external peripheral interrupts are handled by the SIU (see Section 10.4.1, External Interrupt Request Sources).

10.4.1.2 Software Settable Interrupt Requests

The software set/clear interrupt registers (INTC_SSCIR*x*) support the setting or clearing of software-settable interrupt requests. These registers contain eight independent sets of bits to set and clear a corresponding flag bit by software. With the exception of being set by software, this flag bit behaves the same as a flag bit set within a peripheral. This flag bit generates an interrupt request within the INTC just like a peripheral interrupt request.

An interrupt request is triggered by software by writing a 1 to a SET*n* bit in INTC_SSCIR0–INTC_SSCIR7. This write sets a CLR*n* flag bit that generates an interrupt request. The interrupt request is cleared by writing a 1 to the CLR*n* bit. Specific behavior includes the following:

- Writing a 1 to SET*n* leaves SET*n* unchanged at 0 but sets the flag bit (CLR*n* bit).
- Writing a 0 to SET*n* has no effect.
- Writing a 1 to CLR*n* clears the flag (CLR*n*) bit.
- Writing a 0 to CLR*n* has no effect.
- If a 1 is written to a pair of SET*n* and CLR*n* bits at the same time, the flag (CLR*n*) is set, regardless of whether CLR*n* was asserted before the write.

The time from the write to the SET*n* bit to the time that the INTC starts to drive the interrupt request to the processor is four clocks.

10.4.1.3 Unique Vector for Each Interrupt Request Source

Each peripheral and software settable interrupt request is assigned a hardwired unique 9-bit vector. Software settable interrupts 0–7 are assigned vectors 0–7 respectively. The peripheral interrupt requests are assigned vectors 8 to as high as needed to include all the peripheral interrupt requests.

10.4.2 Priority Management

The asserted interrupt requests are compared to each other based on their PRI*n* and PRC_SEL*n* values set in the INTC priority select registers (INTC_PSR0 –INTC_PSR315). The result of that comparison is compared to PRI in the associated current priority register (INTC_CPR_PRC0 or INTC_CPR_PRC1). The results of those comparisons are used to manage the priority of the ISR being executed by the associated processor. The associated LIFO also assists in managing that priority.

10.4.2.1 Current Priority and Preemption

The priority arbitrator, selector, encoder, and comparator submodules shown in Figure 10-1 are used to compare the priority of the asserted interrupt requests to the current priority. If the priority of any asserted peripheral or software settable interrupt request is higher than the current priority for a given processor, then the interrupt request to the processor is asserted. Also, a unique vector for the preempting peripheral or software settable interrupt request is generated for the associated INTC interrupt acknowledge register (INTC_IACKR_PRC0 or INTC_IACKR_PRC1) and, if in hardware vector mode, for the interrupt vector provided to the processor.

10.4.2.1.1 Priority Arbitrator Submodule

The priority arbitrator submodule for each processor compares all the priorities of all of the asserted interrupt requests assigned to that processor, both peripheral and software settable. The output of the priority arbitrator submodule is the highest of those priorities assigned to a given processor. Also, any interrupt requests which have this highest priority are output as asserted interrupt requests to the associated request selector submodule.

10.4.2.1.2 Request Selector Submodule

If only one interrupt request from the associated priority arbitrator submodule is asserted, then it is passed as asserted to the associated vector encoder submodule. If multiple interrupt requests from the associated priority arbitrator submodule are asserted, only the one with the lowest vector passes as asserted to the associated vector encoder submodule. The lower vector is chosen regardless of the time order of the assertions of the peripheral or software settable interrupt requests.

10.4.2.1.3 Vector Encoder Submodule

The vector encoder submodule generates the unique 9-bit vector for the asserted interrupt request from the request selector submodule for the associated processor.

10.4.2.1.4 Priority Comparator Submodule

The priority comparator submodule compares the highest priority output from the associated priority arbitrator submodule with PRI in the associated INTC_CPR_PRC0 or INTC_CPR_PRC1. If the priority comparator submodule detects that the highest priority is higher than the current priority, then it asserts the interrupt request to the associated processor. This interrupt request to the processor asserts whether this highest priority is raised above the value of PRI in the associated INTC_CPR_PRC0 or INTC_CPR_PRC1, or the PRI value in the associated INTC_CPR_PRC0 or INTC_CPR_PRC1 is lowered below this highest priority. This highest priority becomes the new priority which is written to PRI in the associated INTC_CPR_PRC0 or INTC_CPR_PRC1 when the interrupt request to the processor is acknowledged. Interrupt requests whose PRI*n* in INTC_PSR*n*_*n* are 0 will not cause a preemption because their PRI*n* will not be higher than PRI in the associated INTC_CPR_PRC0 or INTC_CPR_PRC1.

Another function of the priority comparator subblock is to signal an update of the INTC_IACKR_PRC0 and INTC_IACKR_PRC1 with the vector number of the first interrupt that arrives that has a priority higher than the current priority. Once the vector number and priority are captured, they cannot be superseded by a higher priority interrupt until an update of the INTC_CPR_PRC0 or INTC_CPR_PRC1 occurs.

10.4.2.2 Last-In First-Out (LIFO)

The LIFO stores the preempted PRI values from the associated INTC_CPR_PRC0 or INTC_CPR_PRC1. Therefore, because these priorities are stacked within the INTC, if interrupts need to be enabled during the ISR, at the beginning of the interrupt exception handler the PRI value in the associated INTC_CPR_PRC0 or INTC_CPR_PRC1 does not need to be loaded from the associated INTC_CPR_PRC0 or INTC_CPR_PRC1 and stored onto the context stack. Likewise, at the end of the interrupt exception handler, the priority does not need to be loaded from the context stack and stored into the associated INTC_CPR_PRC0 or INTC_CPR_PRC1.

The PRI value in the associated INTC_CPR_PRC0 or INTC_CPR_PRC1 is pushed onto the LIFO when the associated INTC_IACKR_PRC0 or INTC_IACKR_PRC1 is read in software vector mode or when the interrupt acknowledge signal from the associated processor is asserted in hardware vector mode. The priority is popped into PRI in the associated INTC_CPR_PRC0 or INTC_CPR_PRC1 when the associated INTC_EOIR_PRC0 or INTC_EOIR_PRC1 is written.

Although the INTC supports 16 priorities, an ISR executing with PRI in the INTC_CPR_PRC0 or INTC_CPR_PRC1 equal to 15 will not be preempted. Therefore, the LIFO supports the stacking of 15 priorities. However, the LIFO is only 14 entries deep. An entry for a priority of 0 is not needed because of how pushing onto a full LIFO and popping an empty LIFO are treated. If the LIFO is pushed 15 or more times than it is popped, the first priorities pushed are overwritten. A priority of 0 would be an overwritten priority. However, the LIFO will pop 0s if it is popped more times than pushed. Therefore, although a priority of 0 was overwritten, it is regenerated with the popping of an empty LIFO.

The LIFO is not memory mapped.

10.4.3 Details on Handshaking with Processor

10.4.3.1 Software Vector Mode Handshaking

10.4.3.1.1 Acknowledging Interrupt Request to Processor

A timing diagram of the interrupt request and acknowledge handshaking in software vector mode, along with the handshake near the end of the interrupt exception handler, is shown in Figure 10-20. The INTC examines the peripheral and software settable interrupt requests. When it finds an asserted peripheral or software settable interrupt request with a higher priority than PRI in the associated INTC current priority register (INTC_CPR_PRC0 or INTC_CPR_PRC1), it asserts the interrupt request to the associated processor. The INTVEC field in the associated INTC current priority register (INTC_IACKR_PRC0 or INTC_IACKR_PRC1) is updated with the preempting interrupt request's vector when the interrupt request to the processor is asserted. The INTVEC field retains that value until the next time the interrupt request to the processor is asserted. The handshaking process is described in Section 10.1.3.1, Software Vector Mode.

10.4.3.1.2 End of Interrupt Exception Handler

Before the interrupt exception handling completes, INTC end-of-interrupt register (INTC_EOIR_PRC0 or INTC_EOIR_PRC1) must be written. When written, the associated LIFO is popped so the preempted priority is restored into PRI of the associated INTC_CPR_PRC0 or INTC_CPR_PRC1. Before it is written, the peripheral or software settable flag bit must be cleared so that the peripheral or software settable interrupt request is negated.

NOTE

To ensure proper operation across all Power Architecture MCUs, execute an **mbar** or **msync** instruction between the access to clear the flag bit and the write to the INTC_EOIR_PRC*n*.

When returning from the preemption, the INTC does not search for the peripheral or software settable interrupt request whose ISR was preempted. Depending on how much the ISR progressed, that interrupt request may no longer be asserted. When PRI in the associated INTC_CPR_PRC0 or INTC_CPR_PRC1 is lowered to the priority of the preempted ISR, the interrupt request for the preempted ISR or other asserted peripherals or software settable interrupt requests at or below that priority will not cause a preemption. Instead, after the restoration of the preempted context, the processor returns to the next instruction address it was about to execute before it was preempted. This next instruction is part of the preempted ISR or the interrupt exception handler's prolog or epilog.

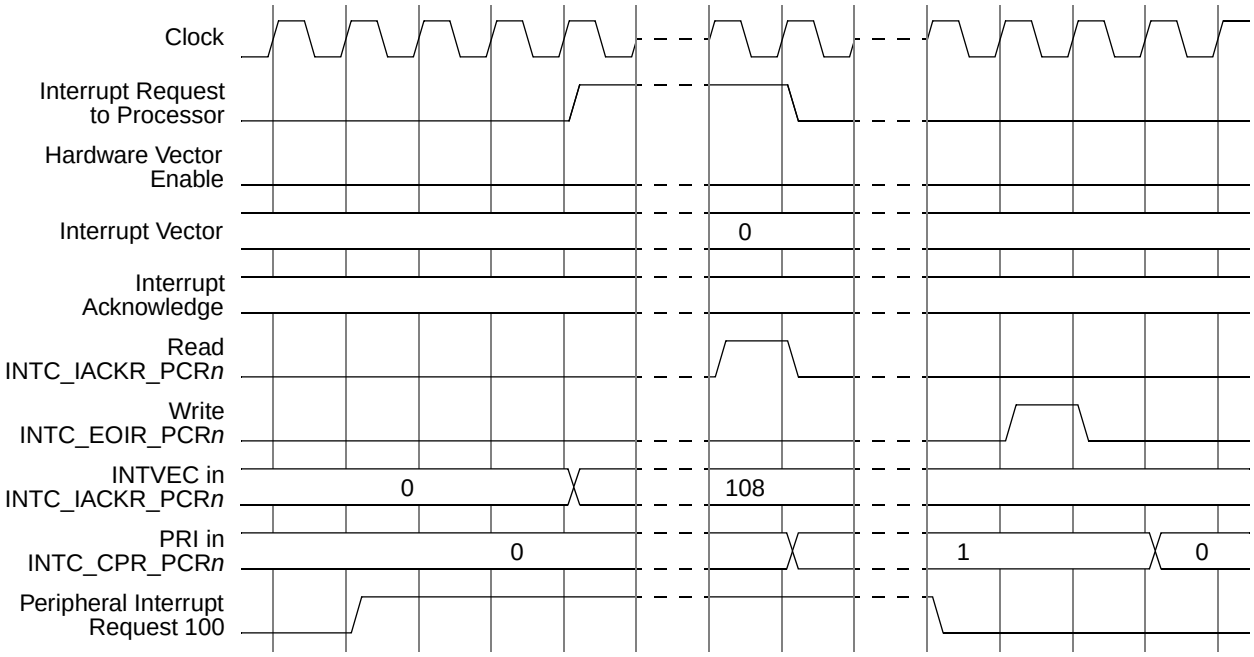


Figure 10-20. Software Vector Mode Handshaking Timing Diagram

10.4.3.2 Hardware Vector Mode Handshaking

A timing diagram of the interrupt request and acknowledge handshaking in hardware vector mode and handshaking near the end of the interrupt exception handler is shown in Figure 10-21. As in software vector mode, the INTC examines the peripheral and software settable interrupt requests and, when it finds one asserted with a higher priority than PRI in the associated INTC_CPR_PRC0 or INTC_CPR_PRC1, it asserts the interrupt request to the associated processor. The INTVEC field in the associated INTC_IACKR_PRC0 or INTC_IACKR_PRC1 is updated with the preempting peripheral or software settable interrupt request's vector when the interrupt request to the processor is asserted. The INTVEC field retains that value until the next time the interrupt request to the associated processor is asserted. In addition, the value of the interrupt vector to the associated processor also matches the value of the INTVEC field in the associated INTC_IACKR_PRC0 or INTC_IACKR_PRC1. The rest of the handshaking process is described in Section 10.1.3.2, Hardware Vector Mode.

The handshaking near the end of the interrupt exception handler, that is written to the associated INTC_EOIR_PRC0 or INTC_EOIR_PRC1, is the same as in software vector mode (see Section 10.4.3.1.2, End of Interrupt Exception Handler).

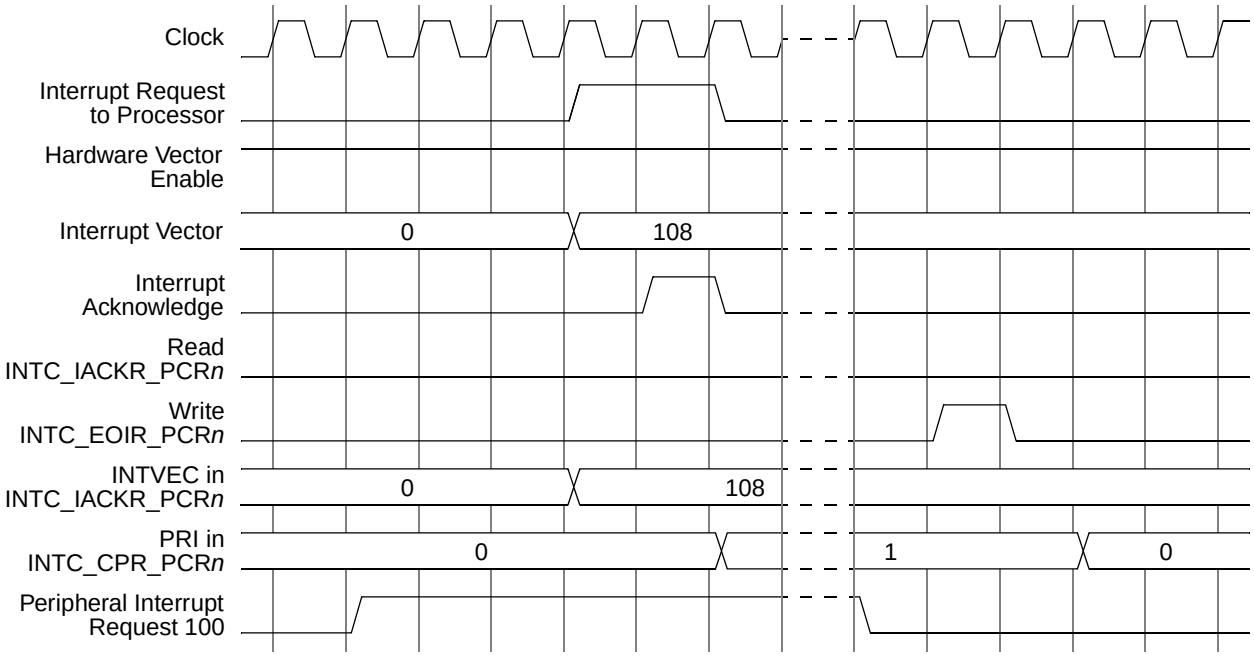


Figure 10-21. Hardware Vector Mode Handshaking Timing Diagram

10.5 Initialization/Application Information

10.5.1 Initialization Flow

After exiting reset, all of the PRI*n* and PRC_SEL*n* fields in the INTC priority select registers (INTC_PSR0–INTC_PSR315) are cleared (set to 0), and PRI in both INTC_CPR_PRC0 and INTC_CPR_PRC1 is set to 0xF (0b1111). These reset values prevent the INTC from asserting interrupt requests to the processors. The enable or mask bits in the peripherals are reset such that the peripheral interrupt requests are negated.

An initialization sequence for allowing the peripheral and software settable interrupt requests to cause an interrupt request to the processor is:

```
interrupt_request_initialization:
configure VTES_PRC0,VTES_PRC1,HVEN_PRC0 and HVEN_PRC1 in INTC_MCR
configure VTBA_PRCn in INTC_IACKR_PRCn
raise the PRIn fields and set the PRC_SELx fields to the desired processor in INTC_PSRn_n
set the enable bits or clear the mask bits for the peripheral interrupt requests
lower PRI in INTC_CPR_PRCn to zero
enable processor(s) recognition of interrupts
```

10.5.2 Interrupt Exception Handler

These example interrupt exception handlers use Power Architecture assembly code.

10.5.2.1 Software Vector Mode

```
interrupt_exception_handler:
code to create stack frame, save working register, and save SRR0 and SRR1

lis      r3,INTC_IACKR_PRCn@ha      # form adjusted upper half of INTC_IACKR_PRCn address
lwz      r3,INTC_IACKR_PRCn@l(r3)   # load INTC_IACKR_PRCn, which clears request to processor
lwz      r3,0x0(r3)                 # load address of ISR from vector table
wrteei   1                          # enable processor recognition of interrupts

code to save rest of context required by e500 EABI

mtlr     r3                         # move address of ISR into link register
blrl                                # branch to ISR; link register updated with epilog
                                    # address

epilog:
code to restore most of context required by e500 EABI

# Popping the LIFO after the restoration of most of the context and the disabling of processor
# recognition of interrupts eases the calculation of the maximum stack depth at the cost of
# postponing the servicing of the next interrupt request.
mbar                                # ensure store to clear flag bit has completed
lis      r3,INTC_EOIR_PRCn@ha       # form adjusted upper half of INTC_EOIR address
li       r4,0x0                     # form 0 to write to INTC_EOIR_PRCn
wrteei   0                          # disable processor recognition of interrupts
stw      r4,INTC_EOIR_PRCn@l(r3)    # store to INTC_EOIR_PRCn, informing INTC to lower priority

code to restore SRR0 and SRR1, restore working registers, and delete stack frame

rfi

vector_table_base_address:
address of ISR for interrupt with vector 0
address of ISR for interrupt with vector 1
         .
         .
         .
address of ISR for interrupt with vector 510
address of ISR for interrupt with vector 511

ISRx:
code to service the interrupt event
code to clear flag bit which drives interrupt request to INTC

blr                                 # return to epilog
```

10.5.2.2 Hardware Vector Mode

This interrupt exception handler is useful with processor and system bus implementations which support a hardware vector. This example assumes that each `interrupt_exception_handlerx` only has space for four instructions, and therefore a branch to `interrupt_exception_handler_continuedx` is needed.

```
interrupt_exception_handlerx:
b        interrupt_exception_handler_continuedx# 4 instructions available, branch to continue

interrupt_exception_handler_continuedx:
```

```
code to create stack frame, save working register, and save SRR0 and SRR1

wrteei   1                          # enable processor recognition of interrupts

code to save rest of context required by e500 EABI

bl       ISRx                       # branch to ISR for interrupt with vector x


epilog:
code to restore most of context required by e500 EABI

# Popping the LIFO after the restoration of most of the context and the disabling of processor
# recognition of interrupts eases the calculation of the maximum stack depth at the cost of
# postponing the servicing of the next interrupt request.
mbar                                # ensure store to clear flag bit has completed
lis      r3,INTC_EOIR_PRCn@ha       # form adjusted upper half of INTC_EOIR_PRCn address
li       r4,0x0                     # form 0 to write to INTC_EOIR_PRCn
wrteei   0                          # disable processor recognition of interrupts
stw      r4,INTC_EOIR_PRCn@l(r3)    # store to INTC_EOIR_PRCn, informing INTC to lower priority

code to restore SRR0 and SRR1, restore working registers, and delete stack frame

rfi


ISRx:
code to service the interrupt event
code to clear flag bit which drives interrupt request to INTC

blr                                 # branch to epilog
```

10.5.3 ISR, RTOS, and Task Hierarchy

The RTOS and all of the tasks under its control typically execute with PRI in INTC current priority register (INTC_CPR_PRC0 or INTC_CPR_PRC1) having a value of 0. The RTOS executes the tasks according to whatever priority scheme it may have, but that priority scheme is independent and has a lower priority of execution than the priority scheme of the INTC. In other words, the ISRs execute above INTC_CPR_PRC*n* priority 0 and outside the control of the RTOS, the RTOS executes at INTC_CPR_PRC*n* priority 0, and while the tasks execute at different priorities under the control of the RTOS, they also execute at INTC_CPR_PRC*n* priority 0.

If a task shares a resource with an ISR and the PCP is being used to manage that shared resource, then the task's priority can be elevated in the INTC_CPR_PRC*n* while the shared resource is being accessed.

An ISR whose PRI*n* in INTC priority select registers (INTC_PSR0–INTC_PSR315) has a value of 0 does not cause an interrupt request to the selected processor, even if its peripheral or software settable interrupt request is asserted. For a peripheral interrupt request, not setting its enable bit or disabling the mask bit causes it to remain negated, which consequently also does not cause an interrupt request to the processor. Since the ISRs are outside the control of the RTOS, this ISR does not run unless called by another ISR or the interrupt exception handler, perhaps after executing another ISR.

10.5.4 Order of Execution

An ISR with a higher priority can preempt an ISR with a lower priority, regardless of the unique vectors associated with each of their peripheral or software settable interrupt requests. However, if multiple peripheral or software settable interrupt requests are asserted, more than one has the highest priority, and that priority is high enough to cause preemption, the INTC selects the one with the lowest unique vector regardless of the order in time that they asserted. However, the ability to meet deadlines with this scheduling scheme is no less than if the ISRs execute in the time order that their peripheral or software settable interrupt requests asserted.

The example in Table 10-13 shows the order of execution of both ISRs with different priorities and the same priority.

Table 10-13. Order of ISR Execution Example

Step#	Step Description	Code Executing at End of Step						PRI in INTC_CPR at End of Step
		RTOS	ISR108[1]	ISR208	ISR308	ISR408	Interrupt Exception Handler	
1	RTOS at priority 0 is executing.	X						0
2	Peripheral interrupt request 100 at priority 1 asserts. Interrupt taken.		X					1
3	Peripheral interrupt request 400 at priority 4 is asserts. Interrupt taken.					X		4
4	Peripheral interrupt request 300 at priority 3 is asserts.					X		4
5	Peripheral interrupt request 200 at priority 3 is asserts.					X		4
6	ISR408 completes. Interrupt exception handler writes to INTC_EOIR_PRC*n*.						X	1
7	Interrupt taken. ISR208 starts to execute, even though peripheral interrupt request 300 asserted first.			X				3
8	ISR208 completes. Interrupt exception handler writes to INTC_EOIR_PRC*n*.						X	1
9	Interrupt taken. ISR308 starts to execute.				X			3
10	ISR308 completes. Interrupt exception handler writes to INTC_EOIR_PRC*n*.						X	1
11	ISR108 completes. Interrupt exception handler writes to INTC_EOIR_PRC*n*.						X	0
12	RTOS continues execution.	X						0

[1] ISR108 executes for peripheral interrupt request 100 because the first eight ISRs are for software settable interrupt requests.

10.5.5 Priority Ceiling Protocol

10.5.5.1 Elevating Priority

The PRI field in INTC current priority register (INTC_CPR_PRC0 or INTC_CPR_PRC1) is elevated in the OSEK PCP to the ceiling of all of the priorities of the ISRs that share a resource. This protocol allows coherent accesses of the ISRs to that shared resource.

For example, ISR1 has a priority of 1, ISR2 has a priority of 2, and ISR3 has a priority of 3. They all share the same resource. Before ISR1 or ISR2 can access that resource, they must raise the PRI value in INTC_CPR_PRC*n* to 3, the ceiling of all of the ISR priorities. After they release the resource, the PRI value in INTC_CPR_PRC*n* can be lowered. If they do not raise their priority, ISR2 can preempt ISR1, and ISR3 can preempt ISR1 or ISR2, possibly corrupting the shared resource. Another possible failure mechanism is deadlock if the higher priority ISR needs the lower priority ISR to release the resource before it can continue, but the lower priority ISR cannot release the resource until the higher priority ISR completes and execution returns to the lower priority ISR.

Using the PCP instead of disabling processor recognition of all interrupts eliminates the time when accessing a shared resource that all higher priority interrupts are blocked. For example, while ISR3 cannot preempt ISR1 while it is accessing the shared resource, all of the ISRs with a priority higher than 3 can preempt ISR1.

10.5.5.2 Ensuring Coherency

Non-coherent accesses to a shared resource can occur. As an example, ISR1 and ISR2 both share a resource. ISR1 has a lower priority, therefore it executes and then writes the new PRI value to the current priority register (INTC_CPR_PRC*n*). The next instruction writes a value to a shared coherent data block.

If INTC asserts the ISR2 interrupt request to the processor just before or at the same time as the first ISR1 write, it is possible for both the ISR1 and ISR2 writes to execute while the processor responds to the INTC request, discards the transactions, and flushes the processing pipeline. However, ISR2 cannot access the data block coherently because the data block is now corrupted.

OSEK uses the GetResource and ReleaseResource system services to manage access to a shared resource. To prevent corrupting a coherent data block, use the following code to modify the PRI in INTC_CPR_PRC*n*. Interrupts must be disabled before executing the following GetResource code sequence:

```
disable processor recognition of interrupts
PRI modification
enable processor recognition of interrupts
```

10.5.5.2.1 Raised Priority Preserved

Before the instruction after the GetResource system service executes, all pending transactions have completed. These pending transactions can include an ISR for a peripheral or software settable interrupt request whose priority was equal to or lower than the raised priority. Also, during the epilog of the interrupt exception handler for this preempting ISR, the raised priority has been restored from the LIFO to PRI in INTC_CPR. The shared coherent data block now can be accessed coherently. Figure 10-22 shows the

timing diagram for this scenario, and Table 10-14 explains the events. The example is for software vector mode, but except for the method of retrieving the vector and acknowledging the interrupt request to the processor, hardware vector mode is identical.

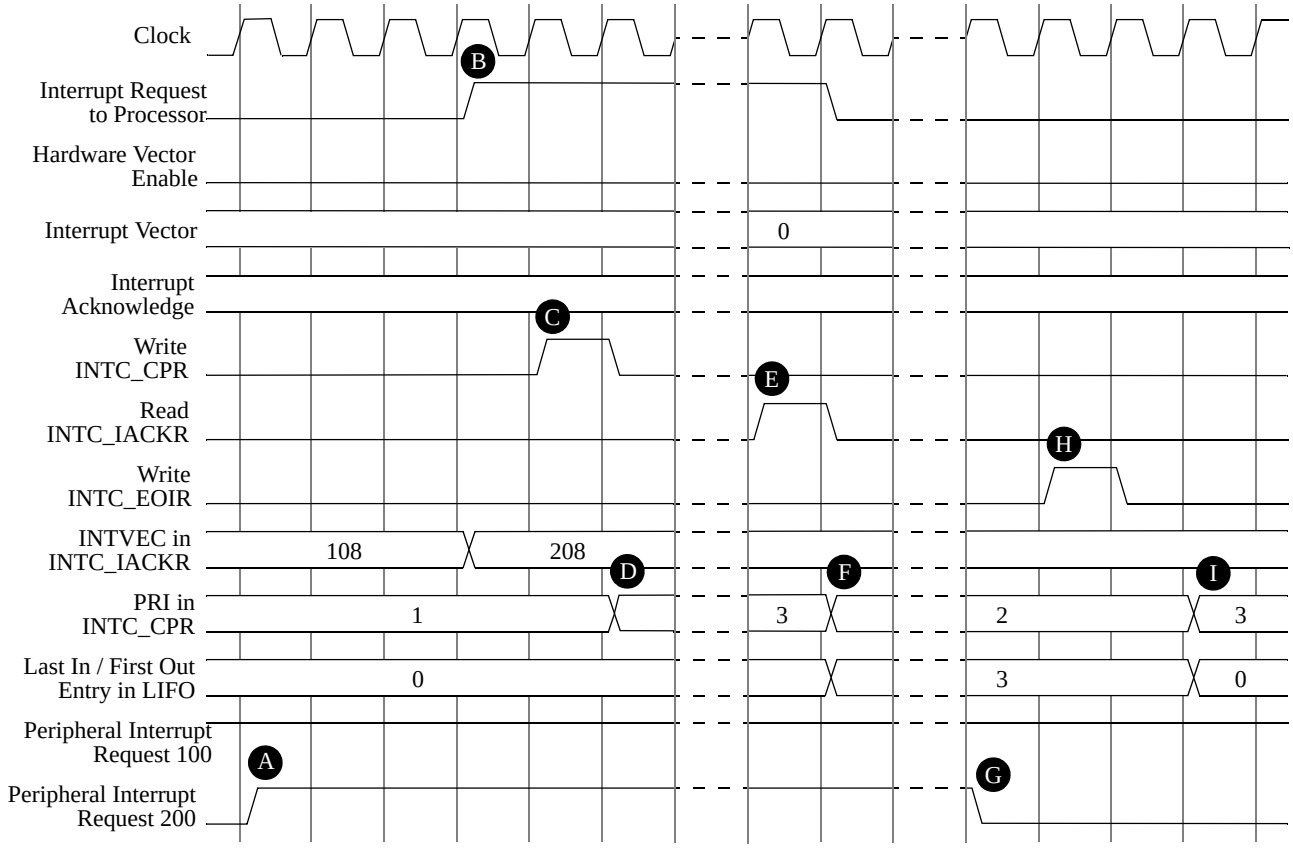


Figure 10-22. Raised Priority Preserved Timing Diagram

Table 10-14. Raised Priority Preserved Events

Event	Description
A	Peripheral interrupt request 200 asserts during execution of ISR108 running at priority 1.
B	Interrupt request to processor asserts. INTVEC in INTC_IACKR updates with vector for that peripheral interrupt request.
C	ISR108 writes to INTC_CPR to raise priority to 3 before accessing shared coherent data block.
D	PRI in INTC_CPR now at 3, reflecting the write. This write, just before accessing data block, is the last instruction the processor executes before being interrupted.
E	Interrupt exception handler prolog acknowledges interrupt by reading INTC_IACKR.
F	PRI of 3 pushed onto LIFO. PRI in INTC_CPR updates to 2, the priority of ISR208.
G	ISR208 clears its flag bit, deasserting its peripheral interrupt request.

Event	Description
H	Interrupt exception handler epilog writes to INTC_EOIR.
I	LIFO pops 3, restoring the raised priority onto PRI in INTC_CPR. Next value to pop from LIFO is the priority from before peripheral interrupt request 100 interrupted. ISR108 now can access data block coherently after interrupt exception handler executes rfi instruction.

10.5.6 Selecting Priorities According to Request Rates and Deadlines

The selection of the priorities for the ISRs can be made using rate monotonic scheduling (RMS) or a superset of it, deadline monotonic scheduling (DMS). In RMS, the ISRs which have higher request rates have higher priorities. In DMS, if the deadline is before the next time the ISR is requested, then the ISR is assigned a priority according to the time from the request for the ISR to the deadline, not from the time of the request for the ISR to the next request for it.

For example, ISR1 executes every 100 µs, ISR2 executes every 200 µs, and ISR3 executes every 300 µs. ISR1 has a higher priority than ISR2 which has a higher priority than ISR3; however, if ISR3 has a deadline of 150 µs, then it has a higher priority than ISR2.

The INTC has 16 priorities, which can be significantly fewer than the number of ISRs. In this case, group the ISRs with other ISRs that have similar deadlines. For example, when a priority is allocated for every time the request rate doubles, ISRs with request rates around 1 ms would share a priority, ISRs with request rates around 500 µs would share a priority, ISRs with request rates around 250 µs would share a priority, etc. With this approach, a range of ISR request rates of 2^{16} could be covered, regardless of the number of ISRs.

Reducing the number of priorities reduces the processor's ability to meet its deadlines. However, it also allows easier management of ISRs with similar deadlines that share a resource. They do not need to use the PCP to access the shared resource.

10.5.7 Software Settable Interrupt Requests

The software settable interrupt requests can be used in two ways. They can be used to schedule a lower priority portion of an ISR and for processors to interrupt other processors in a multiple processor system.

10.5.7.1 Scheduling a Lower Priority Portion of an ISR

A portion of an ISR needs to be executed at the PRI*n* value in INTC priority select registers (INTC_PSR0–INTC_PSR315), which becomes the PRI value in INTC current priority register (INTC_CPR_PRC0 or INTC_CPR_PRC1) with the interrupt acknowledge. The ISR, however, can have a portion of it which does not need to be executed at this higher priority. Therefore, executing this later portion that does not need to be executed at this higher priority can prevent the execution of ISRs that do not have a higher priority than the earlier portion of the ISR but do have a higher priority than what the later portion of the ISR needs. This preemptive scheduling inefficiency reduces the processor's ability to meet its deadlines.

One option is for the ISR to complete the earlier higher priority portion, but then schedule through the RTOS a task to execute the later lower priority portion. However, some RTOSs can require a large amount

of time for an ISR to schedule a task. Therefore, a second option is for the ISR, after completing the higher priority portion, to set a SET*n* bit in INTC software set/clear interrupt registers (INTC_SSCIR0–INTC_SSCIR7). Writing a 1 to SET*n* causes a software settable interrupt request. This software settable interrupt request usually has a lower PRI*n* value in the INTC_PSR*n*, and therefore does not cause preemptive scheduling inefficiencies.

After generating a software settable interrupt request, the higher priority ISR completes. The lower priority ISR is scheduled according to its priority. Execution of the higher priority ISR is not resumed after the completion of the lower priority ISR.

10.5.7.2 Scheduling an ISR on Another Processor

Since the SET*n* bits in the INTC_SSCIR*n* are memory mapped, processors in multiple-processor systems can schedule ISRs on the other processors. One application is that one processor wants to command another processor to perform a piece of work and the initiating processor does not need to use the results of that work. If the initiating processor is concerned that the processor executing the software settable ISR has not completed the work before asking it to again execute the ISR, it can check if the corresponding CLR*n* bit in INTC_SSCIR*n* is asserted before again writing a 1 to the SET*n* bit.

Another application is the sharing of a block of data. For example, a first processor has completed accessing a block of data and wants a second processor to then access it. Furthermore, after the second processor has completed accessing the block of data, the first processor again wants to access it. The accesses to the block of data must be done coherently. The procedure is that the first processor writes a 1 to a SET*n* bit on the second processor. The second processor, after accessing the block of data, clears the corresponding CLR*n* bit and then writes 1 to a SET*n* bit on the first processor, informing it that it now can access the block of data.

10.5.8 Lowering Priority Within an ISR

In implementations without the software-settable interrupt requests in the INTC software set/clear interrupt registers (INTC_SSCIR0–INTC_SSCIR7), one way — besides scheduling a task through an RTOS — to prevent preemptive scheduling inefficiencies with an ISR whose work spans multiple priorities is to lower the current priority (see Section 10.5.7.1, Scheduling a Lower Priority Portion of an ISR). However, the INTC has a LIFO whose depth is determined by the number of priorities.

NOTE

Lowering the PRI value in either INTC_CPR_PRC0 or INTC_CPR_PRC1 within an ISR to below the ISR's corresponding PRI value in INTC_PSR0–INTC_PSR315 allows more preemptions than the LIFO depth can support.

Therefore, through its use of the LIFO, the INTC does not support lowering the current priority within an ISR as a way to avoid preemptive scheduling inefficiencies.

10.5.9 Negating an Interrupt Request Outside of its ISR

10.5.9.1 Negating an Interrupt Request as a Side Effect of an ISR

Some peripherals have flag bits that can be cleared as a side effect of servicing a peripheral interrupt request. For example, reading a specific register can clear the flag bits and consequentially their corresponding interrupt requests too. This clearing as a side effect of servicing a peripheral interrupt request can cause the negation of other peripheral interrupt requests besides the peripheral interrupt request whose ISR presently is executing. This negating of a peripheral interrupt request outside of its ISR can be a desired effect.

10.5.9.2 Negating Multiple Interrupt Requests in One ISR

An ISR can clear other flag bits besides its own. One reason that an ISR clears multiple flag bits is because it serviced those flag bits, and therefore the ISRs for these flag bits do not need to be executed.

10.5.9.3 Proper Setting of Interrupt Request Priority

Whether an interrupt request negates outside its own ISR due to the side effect of an ISR execution or the intentional clearing a flag bit, the priorities of the peripheral or software settable interrupt requests for these other flag bits must be selected properly. Their PRI*n* values in INTC priority select registers (INTC_PSR0–INTC_PSR315) must be selected to be at or lower than the priority of the ISR that cleared their flag bits. Otherwise, those flag bits can cause the interrupt request to the processor to assert. Furthermore, the clearing of these other flag bits also has the same timing relationship to the writing to INTC end-of-interrupt register (INTC_EOIR_PRC*n*) as the clearing of the flag bit that caused the present ISR to be executed (see Section 10.4.3.1.2, End of Interrupt Exception Handler).

A flag bit whose enable bit or mask bit is negating its peripheral interrupt request can be cleared at any time, regardless of the peripheral interrupt request's PRI*n* value in INTC_PSR*x*.

10.5.10 Examining LIFO Contents

Normally you do not need to know the contents of the LIFO, or even how deep the LIFO is nested. Although the LIFO contents are not memory mapped, you can read the contents by popping the LIFO and reading the PRI field in the INTC current priority register (INTC_CPR_PRC0 or INTC_CPR_PRC1). Disabling processor recognition of interrupts while examining the LIFO contents provides a coherent view of the preempted priorities.

The code sequence is:

```
pop_lifo:
store to INTC_EOIR_PRCn
load INTC_CPR_PRCn, examine PRI, and store onto stack
if PRI is not zero or value when interrupts were enabled, branch to pop_lifo
```

When you are finished examining the LIFO contents, you can restore it in software vector mode using the following code sequence:

```
push_lifo:
load stacked PRI value and store to INTC_CPR_PRCn
```

```
load INTC_IACKR_PRCn
if stacked PRI values are not depleted, branch to push_lifo
```

NOTE

Reading the INTC_IACKR_PRC*n* acknowledges the interrupt request to the processor and updates the INTC_CPR_PRC*n*[PRI] with the priority of the preempting interrupt request. If the processor recognition of interrupts is disabled during the LIFO restoration, interrupt requests to the processor can go undetected. However, since the peripheral or software settable interrupt requests are not cleared, the peripheral interrupt request to the processor re-asserts when INTC_CPR_PRC*n*[PRI] is lower than the priorities of those peripheral or software settable interrupt requests.

10.6 Non-Maskable Interrupt (NMI)

The PXN20 can be configured to use the PC6 and PC5 pins as non-maskable interrupts (NMI) by providing a path to the critical interrupt input of the e200z6 and e200z0 cores, respectively.

After the SIU is configured by user code, an NMI cannot be prevented from reaching the assigned core. The only possible way of disabling the critical interrupt is by clearing the critical interrupt enable (CE) bit in the core's machine state register (MSR). The NMI has a higher priority than any interrupt request generated by the INTC, and is not blocked or preempted by any other INTC interrupt request.

After the SIU is properly configured, the operation of the NMI always generates an interrupt request when the programmed edge transition occurs on the pin, regardless of the selected muxing on that pin. It is the user's responsibility to assign pin multiplexing correctly for use with an NMI, which would normally mean selecting it as a port pin rather than a peripheral function.

Figure 10-23 shows the various system level connections needed to create the NMI.

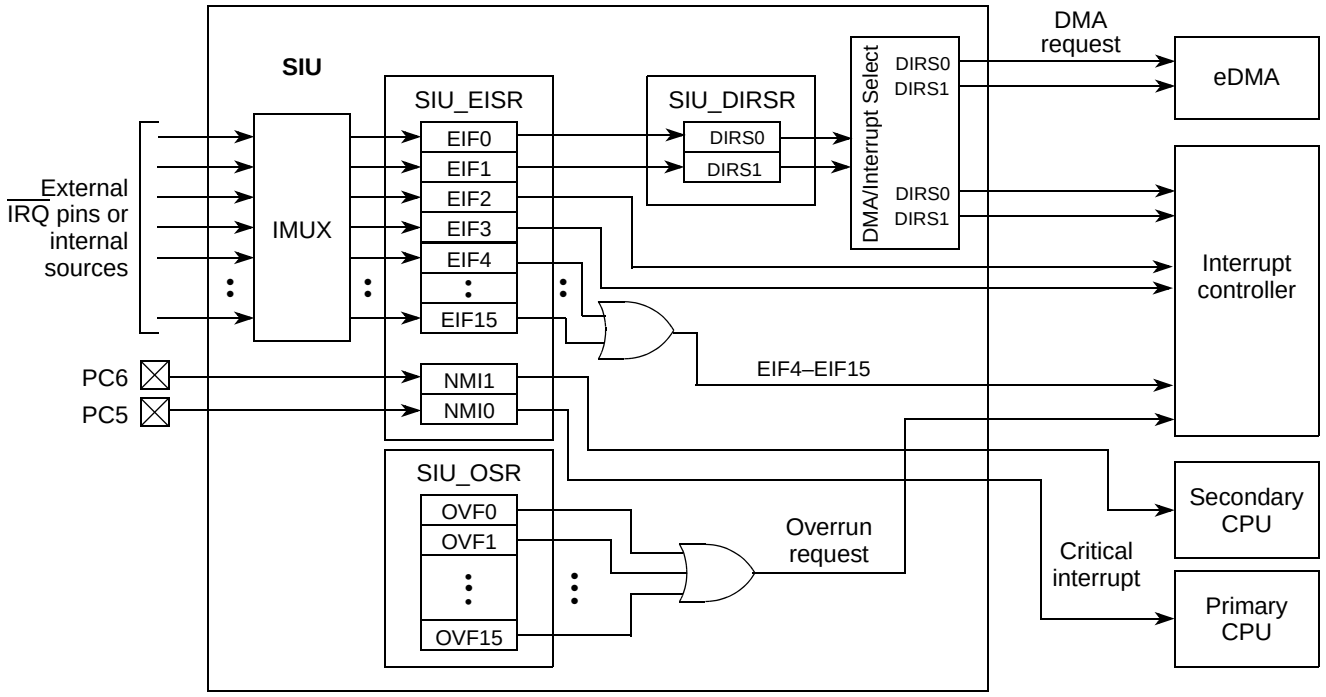


Figure 10-23. NMI Connections

10.7 Dynamic Interrupt Priority Elevation

10.7.1 e200z6 Dynamic Priority Elevation

Dynamic priority elevation is not supported by the e200z6 core since the appropriate control bits are not implemented in the HID1 register.

10.7.2 e200z0 Dynamic Priority Elevation

The e200z0 processor can be configured to support critical and/or external interrupts. Furthermore, the processor can be configured to employ priority elevation on critical and/or external interrupt events. Critical interrupts come from outside the platform, and are routed directly to the processor's critical interrupt input. External interrupts are routed through the interrupt controller. In addition to the interrupt notification signals, various processor specific configuration flags from the e200z0 processor's Machine Check Register (MCR[ee,ce]) and the Hardware Implementation register (HID1) are sent to the ECSM to determine when interrupt servicing is enabled and when high-priority elevation should be enabled. If the e200z0 processor is configured to allow high-priority elevation on critical interrupt events, the ECSM generates the high-priority signal upon critical interrupt detection and holds it active for the duration of interrupt servicing, until a return from critical interrupt (rfci) is detected. If the e200z0 processor is configured to allow high-priority elevation on external interrupt events, the ECSM generates the high-priority signal upon external interrupt detection and holds it active for the duration of interrupt servicing, until a return from interrupt (rfi) is detected.

Great care must be taken when using the priority elevation as it can enable a master to starve the rest of the masters in the system. For more information, see Chapter 19, Error Correction Status Module (ECSM).

10.7.3 eDMA Dynamic Interrupt Priority Elevation

The eDMA can handle dynamic priority elevation via the Bandwidth Control (BWC) field of the transfer control descriptor (TCD). For more information, see Section 24.3.2.17, Transfer Control Descriptor (TCD). The PXN20 DMA multiplexing source allocation is shown in Table 23-4.

Chapter 11
General-Purpose Static RAM (SRAM)

11.1 Introduction

This chapter describes the general-purpose static RAM (SRAM) for the PXN20 family. The PXN20 provides 592 KB of SRAM. The PXN21 provides 128 KB of SRAM.

11.1.1 Block Diagram

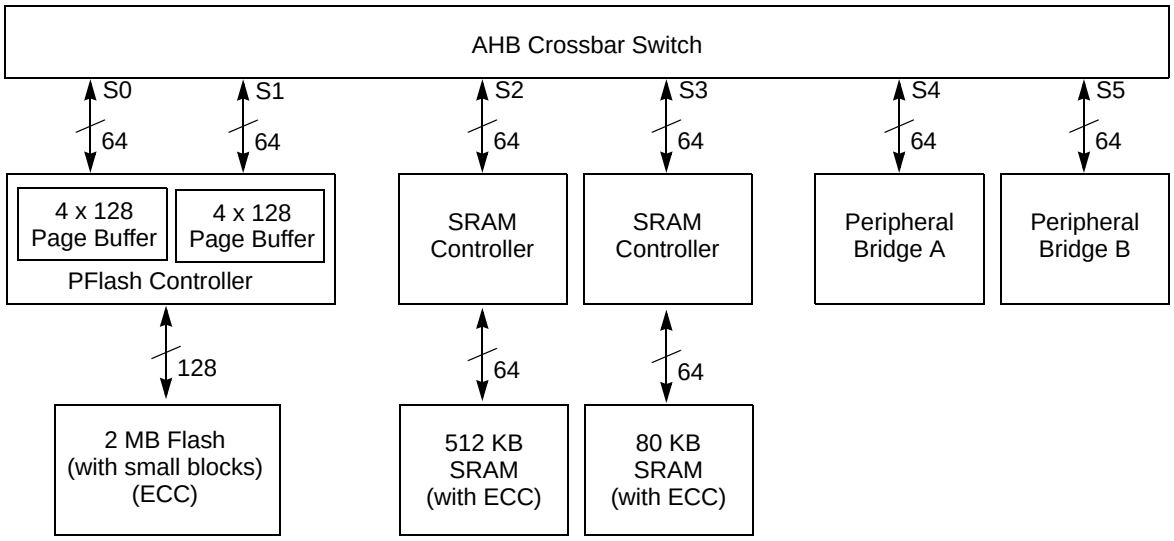


Figure 11-1. Crossbar Arrangement Showing Embedded Memories (PXN20)

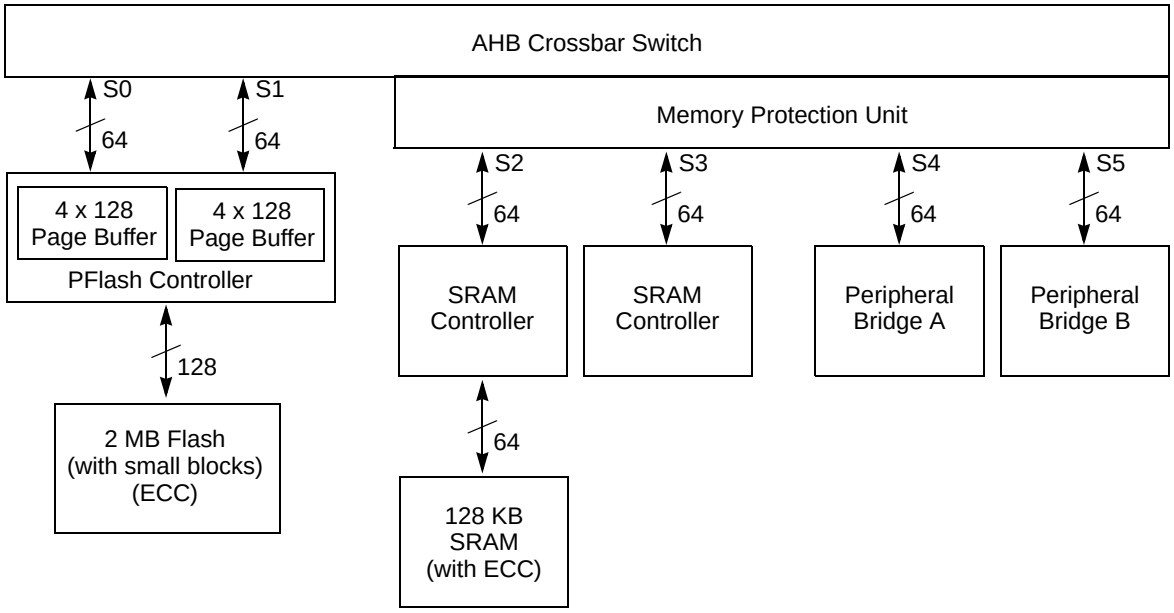


Figure 11-2. Crossbar Arrangement Showing Embedded Memories (PXN21)

11.1.2 Features

Main features of the SRAM module are:

- PXN20
  - — Two separate RAM arrays implemented (592 KB total)
    - – 1 x 80 KB
    - – 1 x 512 KB
- PXN21
  - — One 128 KB RAM array implemented
- 64-bit RAM organization with ECC
- Available for data and program storage
- 64-bit ECC with single-bit correction, double-bit detection on a 32-bit boundary for data integrity
- Supports byte (8-bit), half word (16-bit), word (32-bit) and long word (64-bit) writes for optimal use of memory
- User transparent ECC encoding and decoding for byte, half word, and word accesses
- Separate internal power domains applied RAM block Sleep modes to retain contents during low power mode
- The device can boot from the RAM for fast recovery from low power mode without the need to wait for the Flash to be available.

11.1.3 Modes of Operation

There are two main operating modes of DSPI: normal mode and sleep mode. These modes are briefly described in this section.

11.1.3.1 Normal (Functional) Mode

Normal mode allows for reads and writes of the SRAM memory arrays.

11.1.3.2 Sleep Mode

The size of RAM retained during Sleep mode is controlled in the CRP, in CRP_PSCR[RAMSEL[2:0]]. Sleep mode preserves the contents of the portion of the memory during low-power sleep mode. See Chapter 6, Clocks, Reset, and Power (CRP).

11.2 External Signal Description

There are no external signals associated with the SRAM.

11.3 Memory Map and Registers

There are no control or status registers directly associated with the SRAM module, although error-correcting code (ECC) registers are provided in the error correction status module (ECSM). See Chapter 19, Error Correction Status Module (ECSM), for more information.

The RAM is implemented in two blocks to allow the many masters on the device to access this memory without significantly blocking between the masters. This is necessary since some masters such as the MLB DIM and the FlexRay controller perform significant number of RAM accesses while still allowing the main CPU and IOP to access RAM space.

11.4 Functional Description

ECC checks are performed during the read portion of an SRAM ECC read/write (R/W) operation, and ECC calculations are performed during the write portion of a read/write (R/W) operation. Because the ECC bits can contain random data after the device is powered on, you must initialize the SRAM by executing 64-bit write instructions to the entire SRAM. For more information, refer to Section 11.8, Initialization and Application Information.

11.5 SRAM ECC Mechanism

The SRAM ECC detects the following conditions and produces the following results:

- Detects and corrects all 1-bit errors
- Detects and flags all 2-bit errors as non-correctable errors
- Detects 72-bit reads (64-bit data bus plus the 8-bit ECC) that return all zeros or all ones, asserts an error indicator on the bus cycle, and sets the error flag

The intent of this is to detect all odd-bit failures, all two-bit failures, some three-bit failures, and some multi-bit failures, with regard to IEC 61508-7 A.5.6.

NOTE

The SRAM does not detect all errors greater than 2 bits.

Internal SRAM write operations are performed on the following byte boundaries:

- 1 byte (0:7 bits)
- 2 bytes (0:15 bits)
- 4 bytes or 1 word (0:31 bits)
- 8 bytes or 2 words (0:63 bits)

If the entire 64 data bits are written to SRAM, no read operation is performed and the ECC is calculated across the 64-bit data bus. The 8-bit ECC is appended to the data segment and written to SRAM.

If the write operation is less than the entire 64-bit data width (1-, 2-, or 4-byte segment), the following occurs:

1. The ECC mechanism checks the entire 64-bit data bus for errors, detecting and either correcting or flagging errors.
2. The write data bytes (1-, 2-, or 4-byte segment) are merged with the corrected 64 bits on the data bus.
3. The ECC is then calculated on the resulting 64 bits formed in the previous step.
4. The 8-bit ECC result is appended to the 64 bits from the data bus, and the 72-bit value is then written to SRAM.

11.5.1 Access Timing

The system bus is a two-stage pipelined bus, which makes the timing of any access dependent on the access during the previous clock. Table 14-3 lists the various combinations of read and write operations to SRAM and the number of wait states used for the each operation. The table columns contain the following information:

Current operation	Lists the type of SRAM operation executing currently
Previous operation	Lists the valid types of SRAM operations that can precede the current SRAM operation (valid operation during the preceding clock)
Wait states	Lists the number of wait states (bus clocks) the operation requires which depends on the combination of the current and previous operation

Table 11-1. Number of Wait States Required for SRAM Operations

	Current Operation	Previous Operation	Number of Wait States Required
Read Operation	Read	Idle	1
		Pipelined read	
		Burst read	
		64-bit write	2
		8-, 16-, or 32-bit write	0 (read from the same address)
			1 (read from a different address)
	Pipelined read	Read	0
	Burst read	Idle	1,0,0,0
		Pipelined read	
		Burst read	
		64-bit write	2,0,0,0
		8-, 16-, or 32-bit write	0,0,0,0 (read from the same address)
			1,0,0,0 (read from a different address)
Write Operation	8-, 16-, or 32-bit write	Idle	1
		Read	
		Pipelined 8-, 16-, or 32-bit write	2
		64-bit write	
		8-, 16-, or 32-bit write	0 (write to the same address)
	Pipelined 8-, 16-, or 32-bit write	8-, 16-, or 32-bit write	0
	64-bit write	Idle	0
		64-bit write	
		Read	
	64-bit burst write	Idle	0,0,0,0
		64-bit write	
		Read	

11.5.2 Reset Operation

A destructive reset is associated with an event after which critical register or memory content can no longer be guaranteed, if a write operation occurred during the event.

Destructive resets are as follows.

- Power-on reset
- Low-voltage inhibit (LVI) reset
- External reset
- PLL loss of clock (if enabled)
- PLL loss of lock (if enabled)

The user code must re-initialize the RAM after any of the above resets; otherwise, an ECC event might occur.

11.6 DMA Requests

There are no DMA requests associated with the system SRAM.

11.7 Interrupt Requests

There are no interrupt requests associated with the system SRAM, except for the ECC reporting through the MCM module.

11.8 Initialization and Application Information

To use the SRAM, the ECC must check all bits that require initialization after power on. Use a 64-bit cache-inhibited write to each SRAM location to initialize the SRAM array as part of the application initialization code. All writes must specify an even number of registers performed on 64-bit word-aligned boundaries. If the write is not the entire 64 bits (e.g., 8, 16, or 32 bits), a read / modify / write operation is generated that checks the ECC value upon the read. Refer to Section 11.5, SRAM ECC Mechanism.

NOTE

You *must* initialize SRAM, even if the application does not use ECC reporting.

11.8.1 Example Code

To initialize SRAM correctly, use a store multiple word (**stmw)** instruction to implement 64-bit writes to all SRAM locations. The **stmw** instruction concatenates two 32-bit registers to implement a single 64-bit write. To ensure the writes are 64-bits, specify an even number of registers and write on 64-bit word-aligned boundaries.

The following example code illustrates the use of the **stmw** instruction to initialize the SRAM ECC bits.

Example 11-1. Initializing SRAM ECC Bits

```
init_RAM:
lis      r11,0x4000        # base address of the SRAM, 64-bit word aligned
ori      r11,r11,0         # not needed for this address but could be for others
li       r12,640           # loop counter to get all of SRAM;
                           # 80k/4 bytes/32 GPRs = 640
mtctr    r12
init_ram_loop:
stmw     r0,0(r11)         # write all 32 GPRs to SRAM
```

```
addi      r11,r11,128       # inc the ram ptr; 32 GPRs * 4 bytes = 128?
bdnz      init_ram_loop     # loop for 80k of SRAM
```

Chapter 12
Flash Memory Array and Control

12.1 Introduction

This section presents information about the following components on this device:

- The flash memory block
- The platform flash memory controller

The primary function of the flash memory module is to serve as electrically programmable and erasable non-volatile memory. The NVM memory can be used for instruction and data storage. The block is a non-volatile solid-state silicon memory device consisting of blocks of single-transistor storage elements, an electrical means for selectively adding (programming) and removing (erasing) charge from these elements, and a means of selectively sensing (reading) the charge stored in these elements. The flash is addressable by word (32 bits) and page (128 bits).

The flash block is arranged as two functional units. The first functional unit is the flash core (FC). The FC is composed of arrayed non-volatile storage elements, sense amplifiers, row selects, column selects, charge pumps, and redundancy logic. The arrayed storage elements in the FC are sub-divided into physically separate units referred to as blocks.

The second functional unit of the flash is the memory interface (MI). The MI contains the registers and logic which control the operation of the FC. The MI is also the interface to the flash bus interface unit (FBIU).

The flash core has three address spaces. The low-address space is 256 KB. The mid-address space is also 256 KB. The high-address space is 1.5 MB. The 256 KB of low memory is implemented using eight 16 KB blocks and two 64 KB blocks. The mid-address memory is implemented using two 128 KB blocks. The high memory is implemented using three 512 KB blocks.

Figure 12-1 shows the segmentation for the flash on PXN20.

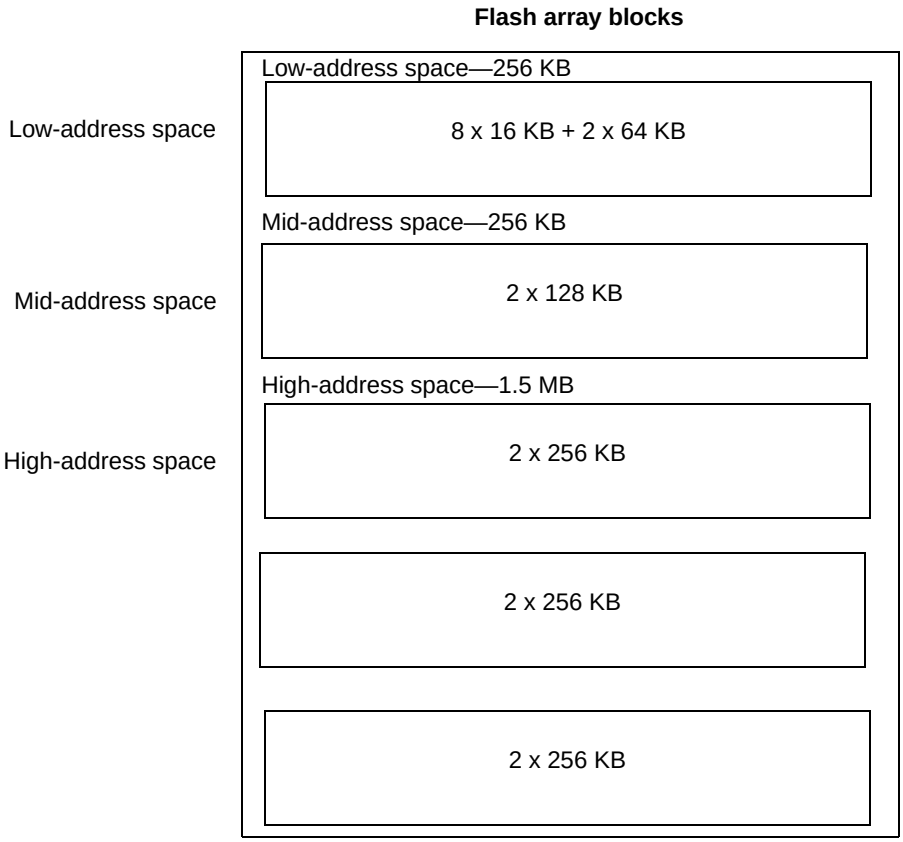


Figure 12-1. Flash Segmentation

12.1.1 Block Diagram

Figure 12-2 shows a block diagram of the flash memory module. The FBIU is addressed through the system bus while the flash control and status registers are addressed through the slave (peripheral) bus.

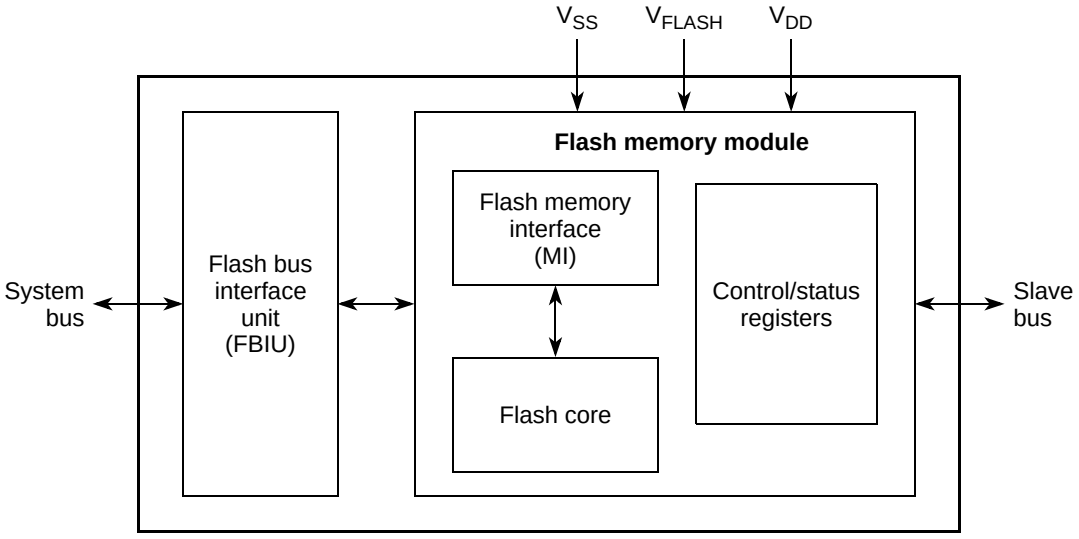


Figure 12-2. Flash System Block Diagram

12.1.2 Features

The flash memory module has these major features:

- Support for a 64-bit data bus for instruction fetch.
- Support for a 32-bit data bus for CPU loads and DMA access. Byte, halfword, word and doubleword reads are supported. Only aligned word and doubleword writes are supported.
- Configurable read buffering and line prefetch support. Four line read buffers (128 bits wide) and a prefetch controller are used to support single-cycle read responses for hits in the buffers.
- Hardware and software configurable read and write access protections on a per-master basis.
- Interface to the flash array controller is pipelined with a depth of 1, allowing overlapped accesses to proceed in parallel for interleaved or pipelined flash array designs.
- Configurable access timing allowing use in a wide range of system frequencies.
- Multiple-mapping support and mapping-based block access timing (0-31 additional cycles) allowing use for emulation of other memory types.
- Software programmable block program/erase restriction control for low, mid and high address spaces.
- Erase of selected block(s).
- Read page size of 128 bits (4 words).
- ECC with single-bit correction, double-bit detection.
- Minimum program size is 2 consecutive 32 bit words, aligned on a 0-modulo-8 byte address, due to ECC.
- Embedded hardware program and erase algorithm.
- Read While Write (RWW) with multiple partitions.
- Sleep mode for low power stand-by.
- Erase suspend, program suspend and erase-suspended program.
- Automotive flash which meets automotive endurance and reliability requirements.
- Shadow information stored in non-volatile shadow block.
- Independent program/erase of the shadow block.

12.1.3 Modes of Operation

12.1.3.1 Flash User Mode

User mode is the default operating mode of the flash module. In this mode, it is possible to read and write, program and erase the flash module.

12.1.3.2 Sleep Mode

Sleep mode turns off most DC current sources within the module. The module is not accessible for read or write once put to sleep.

12.1.3.3 User Test Mode (UTest)

User test mode (UTest) provides a limited set of tests to end users.

12.2 External Signal Description

V_{DD} is the only externally visible power supply that is necessary for programming and erasing the flash array. The other flash supplies are tied to the appropriate supply pads in the package. Refer to Section 3.4.15, Power / Ground Signals, and the *PXN20 Microcontroller Data Sheet*.

12.3 Memory Map and Registers

This section provides a detailed description of all flash memory registers.

12.3.1 Module Memory Map

The flash memory map is shown in Table 12-1. The addresses are given as an offset to the flash memory base address.

The flash register memory map is shown in Table 12-2. There are no program-visible registers that physically reside inside the flash. The flash receives control and configuration information from the flash array controller to determine operating configurations. These are part of the flash array controller's configuration registers mapped into the IPS address space but are described herein. These registers should only be referenced with 32-bit accesses.

Table 12-1. Flash Memory Map

Offset from FLASH_BASE (0x0000_0000)	Use	Block[1]	Partition
0x0000_0000	Low-address space	L0	1
0x0000_4000		L1	
0x0000_8000		L2	
0x0000_C000		L3	
0x0001_0000		L4	2
0x0001_4000		L5	
0x0001_8000		L6	
0x0001_C000		L7	
0x0002_0000		L8	3
0x0003_0000		L9	
0x0004_0000	Mid-address space	M0	4
0x0006_0000		M1	

Table 12-1. Flash Memory Map (continued)

Offset from FLASH_BASE (0x0000_0000)	Use	Block[1]	Partition
0x0008_0000	High-address space	H0	5
0x000C_0000		H1	
0x0010_0000		H2	6
0x0014_0000		H3	
0x0018_0000		H4	7
0x001C_0000		H5	
0x0020_0000–0x00FF_BFFF	Reserved		
0x00FF_C000–0x00FF_FDD7	General use	S	All[2]
0x00FF_FDD8	Serial passcode (0xFEED_FACE_CAFE_BEEF)		
0x00FF_FDE0	Censorship control word (0x55AA_55AA)		
0x00FF_FDE4	General use		
0x00FF_FDE8	LML reset configuration (0x0010_0000)		
0x00FF_FDEC	General use		
0x00FF_FDF0	HBL reset configuration (0x0FFF_FFFF)		
0x00FF_FDF4	General use		
0x00FF_FDF8	SLL reset configuration (0x000F_FFFF)		
0x00FF_FDFC–0x00FF_FFFF	General use		

1 L*n* = Low Address Space, M*n* = Mid Address Space, H*n* = High Address Space, S = Shadow Block.

2 For read while write operations, the shadow row behaves as if it is in all partitions.

Table 12-2. Flash Configuration Register Memory Map

Offset from FLASH_REGS_BASE (0xFFFF_8000)	Register	Access	Reset Value	Section/Page
0x0000	MCR—Module configuration register	R/W[1]	0x0540_0600	12.3.2.1/12-6
0x0004	LML—Low-/Mid-address space block locking register	R/W[1]	0x0013_03FF	12.3.2.2/12-10
0x0008	HBL—High-address space block locking register	R/W[1]	0x0000_003F	12.3.2.3/12-12
0x000C	SLL—Secondary low-/mid-address space block locking register	R/W[1]	0x0013_03FF	12.3.2.4/12-13
0x0010	LMS—Low-/mid-address space block select register	R/W[1]	0x0000_0000	12.3.2.5/12-14
0x0014	HBS—High-address space block select register	R/W[1]	0x0000_0000	12.3.2.6/12-15
0x0018	ADR—Address register	R/W[1]	0x0000_0000	12.3.2.7/12-16
0x001C	PFCRP0—Platform flash configuration register for port 0	R/W[1]	0x0800_FF00	12.3.2.8/12-17
0x0020	PFCRP1—Platform flash configuration register for port 1	R/W[1]	0x3000_FF00	12.3.2.8/12-17
0x0024	PFAPR—Platform flash access protection register	R/W	0x00FF_FE00	12.3.2.9/12-20

Table 12-2. Flash Configuration Register Memory Map (continued)

Offset from FLASH_REGS_BASE (0xFFFF_8000)	Register	Access	Reset Value	Section/Page
0x0028	PFSACC—Platform flash supervisor access control register	R/W[1]	0x00FF_FE08	12.3.2.10/12-21
0x002C	PFDACC—Platform flash data access control register	R/W[1]	0x00FF_FE10	12.3.2.11/12-23
0x0030 – 0x0038	Reserved			
0x003C	UT0—UTest register 0	R/W[1]	0x0000_0001	12.3.2.12/12-23
0x0040	UT0—UTest register 1	R/W	0x0000_0000	12.3.2.13/12-25
0x0044	UT0—UTest register 2	R/W	0x0000_0000	12.3.2.14/12-26
0x0048	UM0—User multiple input signature register 0	R/W	0x0000_0000	12.3.2.15/12-26
0x004C	UM1—User multiple input signature register 1	R/W	0x0000_0000	12.3.2.15/12-26
0x0050	UM2—User multiple input signature register 2	R/W	0x0000_0000	12.3.2.15/12-26
0x0054	UM3—User multiple input signature register 3	R/W	0x0000_0000	12.3.2.15/12-26
0x0058	UM4—User multiple input signature register 4	R/W	0x0000_0000	12.3.2.15/12-26
0x0048 – 0x3FFF	Reserved			

1 Some bits are read-only.

12.3.2 Register Descriptions

This section lists the flash memory registers in address order and describes the registers and their bit fields.

12.3.2.1 Module Configuration Register (MCR)

The MCR register is shown in Figure 12-3 and Table 12-3.

Offset: FLASH_REGS_BASE + 0x0000 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	SIZE			0	LAS			0	0	0	MAS
W																
Reset	0	0	0	0	0	1	0	1	0	1	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	EER	RWE	SBC	0	PEAS	DONE	PEG	0	0	0	0	PGM	PSUS	ERS	ESUS	EHV
W	w1c	w1c	w1c													
Reset	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0

Figure 12-3. Module Configuration Register (MCR)

Table 12-3. MCR Field Descriptions

Field	Description
SIZE	Array Space Size. The value of the SIZE field depends on the size of the flash module. For PXN20, this bit field is 0b101, indicating a 2.0 MB array size (with 1.5 MB in high-address space). SIZE is read only.
LAS	Low Address Space. The value of the LAS field corresponds to the configuration of the Low Address Space. For PXN20, this bit field is 0b100, indicating eight 16 KB blocks and two 64 KB blocks. LAS is read only.
MAS	Mid Address Space. The value of the MAS field corresponds to the configuration of the Mid Address Space. The value of the MAS field depends on the size of the flash module. For the PXN20, this bit field is 0b0, indicating two 128 KB blocks. MAS is read only.
EER	ECC Event Error. EER provides information on previous reads. If a double bit detection occurred, the EER bit is set to a 1. This bit must then be cleared, or a reset must occur before this bit returns to a 0 state. This bit may not be set by the user. In the event of a single bit detection and correction, this bit is not be set. If EER is not set, or remains 0, this indicates that all previous reads (from the last reset, or clearing of EER) are correct. Since this bit is an error flag, it must be cleared to a 0 by writing a 1 to the register location. A write of 0 has no effect. 0 Reads are occurring normally. 1 An ECC Error occurred during a previous read.
RWE	Read While Write Event Error. RWE provides information on previous RWW reads. If a Read While Write error occurs, this bit is set to 1. This bit must then be cleared, or a reset must occur before this bit returns to a 0 state. This bit may not be written to a 1 by the user. If RWE is not set, or remains 0, this indicates that all previous RWW reads (from the last reset, or clearing of RWE) are correct. Since this bit is an error flag, it must be cleared to a 0 by writing a 1 to the register location. A write of 0 has no effect. 0 Reads are occurring normally. 1 A Read While Write Error occurred during a previous read.
SBC	Single Bit Correction. SBC provides information on previous reads provided the UT0[SPCE] is set. If a single bit correction occurred, the SBC bit is set to a 1. This bit must then be cleared, or a reset must occur before this bit returns to a 0 state. If SBC is not set, or remains 0, this indicates that all previous reads (from the last reset, or clearing of SBC) did not require a correction. Since this bit is an error flag, it must be cleared to a 0 by writing a 1 to the register location. A write of 0 has no effect. 0 Reads are occurring without corrections. 1 A Single Bit Correction occurred during a previous read.
PEAS	Program/Erase Access Space. PEAS is used to indicate which space is valid for program and erase operations, either main array space or shadow space. PEAS = 0 indicates that the main address space is active for all FC program and erase operations. PEAS = 1 indicates the shadow address space is active for program/erase. The value in PEAS is captured and held when the shadow block is enabled with the first interlock write done for program or erase operations. The value of PEAS is retained between sampling events (i.e.,subsequent first interlock writes). The value in PEAS may be changed during erase-suspended program, and reverts back to its' original state once the erase-suspended program is completed. PEAS is read only. 0 Shadow address space is disabled for program/erase and main address space enabled. 1 Shadow address space is enabled for program/erase and main address space disabled.
DONE	State Machine Status. Indicates if the flash module is performing a high-voltage operation. DONE is set to a 1 on termination of the flash module reset, at the end of program and erase high-voltage sequences and after a successful abort of a high voltage operation. DONE is cleared upon commencement of a high voltage operation or on the resumption of a suspended operation. 0 Flash is executing a high-voltage operation 1 Flash is not executing a high-voltage operation

Table 12-3. MCR Field Descriptions (continued)

Field	Description
PEG	Program/Erase Good. The PEG bit indicates the completion status of the last flash program or erase sequence for which high voltage operations were initiated. The value of PEG is updated automatically during the program and erase high voltage operations. Aborting a program/erase high voltage operation causes PEG to be cleared, indicating the sequence failed. PEG is set to a 1 when the module is reset. PEG is read only. The value of PEG is valid only when PGM = 1 and/or ERS = 1 and after DONE transitions from 0 to 1 due to an abort or the completion of a program/erase operation. PEG is valid until PGM/ERS makes a 1 to 0 transition or EHV makes a 0 to 1 transition. The value in PEG is not valid after a 0 to 1 transition of DONE caused by PSUS or ESUS being set to logic 1. If PGM and ERS are both 1 when DONE makes a qualifying 0 to 1 transition the value of PEG indicates the completion status of the PGM sequence. This happens in an erase-suspended program operation. 0 Program or erase operation failed. 1 Program or erase operation successful. **Note:** If program or erases are attempted on blocks that are locked, the response from flash is PEG = 1, indicating that the operation was successful, and the contents of the block are properly protected from the program or erase operation.
PGM	Program. PGM is used to set up flash for a program operation. A 0 to 1 transition of PGM initiates a program sequence. A 1 to 0 transition of PGM ends the program sequence. PGM can be set only under one of the following conditions: • User mode read (ERS is low and UTE is low). • Erase suspend (ERS and ESUS are 1) with EHV low. PGM can be cleared by the user only when PSUS and EHV are low and DONE is high. PGM is cleared on reset. 0 Flash is not executing a program sequence. 1 Flash is executing a program sequence. **Note:** In an erase-suspended program, programming Flash locations in blocks which were being operated on in the erase may corrupt FC data. This should be avoided due to reliability implications.
PSUS	Program Suspend. PSUS is used to indicate the flash module is in program suspend or in the process of entering a suspend state. The module is in program suspend when PSUS = 1 and DONE = 1. PSUS can be set high only when PGM and EHV are high. A 0 to 1 transition of PSUS starts the sequence which sets DONE and places the flash module in program suspend. PSUS can be cleared only when DONE and EHV are high. A 1 to 0 transition of PSUS with EHV = 1 starts the sequence which clears DONE and returns the flash module to program. The module cannot exit program suspend and clear DONE while EHV is low. PSUS is cleared on reset. 0 Program sequence is not suspended. 1 Program sequence is suspended.
ERS	Erase. ERS is used to set up flash for an erase operation. A 0 to 1 transition of ERS initiates an erase sequence. A 1 to 0 transition of ERS ends the erase sequence. ERS can only be set only in user mode read (PGM is low and UTE is low). ERS can be cleared by the user only when ESUS and EHV are low and DONE is high. ERS is cleared on reset. 0 Flash is not executing an erase sequence. 1 Flash is executing an erase sequence.

Table 12-3. MCR Field Descriptions (continued)

Field	Description
ESUS	Erase Suspend. ESUS is used to indicate that the flash module is in erase suspend or in the process of entering a suspend state. The module is in erase suspend when ESUS = 1 and DONE = 1. ESUS can be set high only when ERS and EHV are high and PGM is low. A 0 to 1 transition of ESUS starts the sequence which sets DONE and places the flash in erase suspend. ESUS can be cleared only when DONE and EHV are high and PGM is low. A 1 to 0 transition of ESUS with EHV = 1 starts the sequence which clears DONE and returns the module to erase. The flash module cannot exit erase suspend and clear DONE while EHV is low. ESUS is cleared on reset. 0 Erase sequence is not suspended. 1 Erase sequence is suspended.
EHV	Enable High Voltage. The EHV bit enables the flash module for a high voltage program/erase operation. EHV is cleared on reset. EHV must be set after an interlock write to start a program/erase sequence. EHV may be set, initiating a program/erase, after an interlock under one of the following conditions: • Erase (ERS = 1, ESUS = 0). • Program (ERS = 0, ESUS = 0, PGM = 1, PSUS = 0). • Erase-suspended program (ERS = 1, ESUS = 1, PGM = 1, PSUS = 0). If a program operation is to be initiated while an erase is suspended the user must clear EHV while in erase suspend before setting PGM. In normal operation, a 1 to 0 transition of EHV with DONE high, PSUS and ESUS low terminates the current program/erase high voltage operation. When an operation is aborted, there is a 1 to 0 transition of EHV with DONE low and the suspend bit for the current program/erase sequence low. An abort causes the value of PEG to be cleared, indicating a failed program/erase; address locations being operated on by the aborted operation contain indeterminate data after an abort. A suspended operation cannot be aborted. EHV may be written during suspend. EHV must be high for the flash module to exit suspend. EHV may not be written after a suspend bit is set high and before DONE transitions high. EHV may not be set low after the current suspend bit is set low and before DONE transitions low. 0 Flash is not enabled to perform a high voltage operation. 1 Flash is enabled to perform a high voltage operation. **Note:** Aborting a high voltage operation leaves FC addresses in an indeterminate data state. This may be recovered by executing an erase on the affected blocks.

NOTE

The program and erase sequence of the flash may be suspended to allow read and program access to the flash core. A suspend operation is initiated by setting the Erase Suspend (ESUS) bit or Program Suspend (PSUS) bit in the flash Module Configuration Register (MCR). Setting a suspend bit causes the flash module to start the sequence, which places it in the suspended state. The user must then wait until the MCR[DONE] bit is set before a read or program to the flash is initiated, as the high voltage operation needs to be complete to avoid errors. However, during normal read to the same partition, following a suspend sequence, (setting MCR bit and waiting for MCR[DONE] bit to be set) can result in read fails that will return multiple bit ECC errors. The error is due to the MCR[DONE] bit being set before the internal high voltage operation is complete.

Because the MCR[DONE] flag can be set too soon, a delay needs to be inserted between setting the MCR[ESUS] or MCR[PSUS] and reading the same flash partition. The minimum duration of the delay should be 40 us to guarantee correct operation.

12.3.2.1.1 MCR Simultaneous Register Writes

A number of MCR bits are protected against write when another bit, or set of bits, is in a specific state. These write locks are covered on a bit by bit basis in the preceding section. The write locks detailed in the previous section do not consider the effects of trying to write two or more bits simultaneously. The effects of writing bits simultaneously which put the module in an illegal state are detailed here.

The flash module does not allow the user to write bits simultaneously which put the device into an illegal state. This is implemented through a priority mechanism among the bits. The bit changing priorities are detailed in Table 12-4.

Table 12-4. MCR Bit Set/Clear Priority Levels

Priority Level	MCR Bit(s)
1	ERS
2	PGM
3	EHV
4	ESUS, PSUS

If the user attempts to write two or more MCR bits simultaneously then only the bit with the lowest priority level is written. Setting two bits with the same priority level is prevented by existing write locks or do not put the flash in an illegal state.

For example, setting ERS and PGM simultaneously results in only ERS being set. Attempting to clear EHV while setting PSUS results in EHV being cleared, while PSUS is unaffected.

12.3.2.2 Low/Mid Address Space Block Locking Register (LML)

The Low/Mid Address Block Locking Register (LML) provides a means to protect blocks from being modified. These bits, along with bits in the Secondary LLOCK (SLL), determine if the block is locked from program or erase. An “OR” of LML and SLL determine the final lock status.

NOTE

A reset value of 1* in Figure 12-4 indicates that the reset value of these registers is determined by Flash values in the shadow block. An erased shadow block causes the reset value to be 1.

The LML register is shown in Figure 12-4 and Table 12-5.

Offset: FLASH_REGS_BASE + 0x0004 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	LME	0	0	0	0	0	0	0	0	0	0	SLOCK	0	0	MLOCK	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	1*	0	0	1*	1*

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	LLOCK									
W																
Reset	0	0	0	0	0	0	1*	1*	1*	1*	1*	1*	1*	1*	1*	1*

Figure 12-4. Low/Mid Address Block Locking Register (LML)

Table 12-5. LML Field Descriptions

Field	Description
LME	Low/Mid Address Lock Enable. This bit is used to enable the Lock registers (SLOCK, MLOCK and LLOCK) to be set or cleared by register writes. This bit is a status bit only, and may not be written or cleared, and the reset value is 0. The method to set this bit is to write a password, and if the password matches, the LME bit is set to reflect the status of enabled, and is enabled until a reset operation occurs. For LME, the password 0xA1A1_1111 must be written to the LML register. 0 Low/Mid Address Locks are disabled, and cannot be modified. 1 Low/Mid Address Locks are enabled to be written.
SLOCK	Shadow Lock. This bit is used to lock the shadow block from programs and erases. The SLOCK register is not writable once an interlock write is completed until MCR[DONE] is set at the completion of the requested operation. Likewise, SLOCK register is not writable if a high voltage operation is suspended. SLOCK is also not writeable during UTest operations, when AIE is high. Upon reset, information from the shadow block is loaded into the SLOCK register. The SLOCK bit may be written as a register. Reset causes the bits to go back to their shadow block value. The default value of the SLOCK bits (assuming erased shadow location) is locked. SLOCK is not writable unless LME is high. 0 The shadow block can receive program and erase pulses. 1 The shadow block is locked for program and erase.

Table 12-5. LML Field Descriptions (continued)

Field	Description
MLOCK[1:0]	Mid Address Space Block Lock. A value of 1 in a bit of the lock register signifies that the corresponding block is locked for program and erase. A value of 0 in the lock register signifies that the corresponding block is available to receive program and erase pulses. The block numbering for Mid Address Space starts with MLOCK[0] and continues until all blocks are accounted. The lock register is not writable once an interlock write is completed until MCR[DONE] is set at the completion of the requested operation. Likewise, the lock register is not writable if a high voltage operation is suspended. MLOCK is also not writeable during UTest operations, when AIE is high. Upon reset, information from the shadow block is loaded into the block registers. The LOCK bits may be written as a register. Reset causes the bits to go back to their shadow block value. The default value of the LOCK bits (assuming erased shadow location) is locked. In the event that blocks are not present (due to configuration or total memory size), the LOCK bits default to be locked, and are not writable. The reset value is always 1 (independent of the shadow block), and register writes have no effect. MLOCK is not writable unless LME is high.
LLOCK[9:0]	Low Address Space Block Lock. A value of 1 in a bit of the lock register signifies that the corresponding block is locked for program and erase. A value of 0 in the lock register signifies that the corresponding block is available to receive program and erase pulses. The block numbering for Low Address Space starts with LLOCK[0] and continues until all blocks are accounted. For more details on LLOCK, please see MLOCK bit description. LLOCK is not writable unless LME is high.

12.3.2.3 High Address Space Block Locking Register (HBL)

The High Address Space Block Locking Register (HBL) provides a means to protect blocks from being modified.

NOTE

A reset value of 1* in Figure 12-5 indicates that the reset value of these registers is determined by Flash values in the shadow block. An erased shadow block causes the reset value to be 1.

The HBL register is shown in Figure 12-5 and Table 12-6.

Offset: FLASH_REGS_BASE + 0x0008 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	HBE	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	HLOCK					
W																
Reset	0	0	0	0	0	0	0	0	0	0	1*	1*	1*	1*	1*	1*

Figure 12-5. High Address Space Block Locking Register (HBL)

Table 12-6. HBL Field Descriptions

Field	Description
HBE	High Address Lock Enable This bit is used to enable the Lock registers (HLOCK) to be set or cleared by register writes. This bit is a status bit only, and may not be written or cleared, and the reset value is 0. The method to set this bit is to provide a password, and if the password matches, the HBE bit is set to reflect the status of enabled, and is enabled until a reset operation occurs. For HBE, the password 0xB2B2_2222 must be written to the HBL register. 0 High Address Locks are disabled, and cannot be modified. 1 High Address Locks are enabled to be written.
HLOCK[5:0]	High Address Space Block Lock. HLOCK has the same characteristics as LLOCK. Please see this description for more information. The block numbering for High Address Space starts with HLOCK[0] and continues until all blocks are accounted. HLOCK is not writable unless HBE is high.

12.3.2.4 Secondary Low/Mid Address Space Block Locking Register (SLL)

The Secondary Low/Mid Address Block Locking Register (SLL) provides an alternative means to protect blocks from being modified. This has the effect of creating a “tiered” locking scheme to enable different flash users to provide different default locking on blocks. These bits, along with bits in the LLOCK (LML), determine if the block is locked from program or erase. An “OR” of LML and SLL determine the final lock status.

NOTE

A reset value of 1* in Figure 12-6 indicates that the reset value of these registers is determined by Flash values in the shadow block. An erased shadow block causes the reset value to be 1.

The SLL register is shown in Figure 12-6 and Table 12-7.

Offset: FLASH_REGS_BASE + 0x000C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	SLE	0	0	0	0	0	0	0	0	0	0	SS LOCK	0	0	SM LOCK	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	1*	0	0	1*	1*

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	SLLOCK									
W																
Reset	0	0	0	0	0	0	1*	1*	1*	1*	1*	1*	1*	1*	1*	1*

Figure 12-6. Secondary Low/Mid Address Block Locking Register (SLL)

Table 12-7. SLL Field Descriptions

Field	Description
SLE	Secondary Low/Mid Address Lock Enable. This bit is used to enable the Lock registers (SSLOCK, SMLOCK, and SLLOCK) to be set or cleared by register writes. This bit is a status bit only, and may not be written or cleared, and the reset value is 0. The method to set this bit is to provide a password, and if the password matches, the SLE bit is set to reflect the status of enabled, and is enabled until a reset operation occurs. For SLE, the password 0xC3C3_3333 must be written to the SLL register 0 Secondary Low/Mid Address Locks are disabled, and cannot be modified. 1 Secondary Low/Mid Address Locks are enabled to be written.
SSLOCK	Secondary Shadow Lock. This bit is an alternative method that may be used to lock the shadow block from programs and erases. SSLOCK has the same description as SLOCK. SSLOCK is not writable unless SLE is high.
SMLOCK[1:0]	Secondary Mid Address Block Lock. This bit is an alternative method that may be used to lock the Mid Address Space blocks from programs and erases. SMLOCK has the same description as MLOCK. SMLOCK is not writable unless SLE is high.
SLLOCK[9:0]	Secondary Low Address Block Lock. This bit is an alternative method that may be used to lock the Low Address Space blocks from programs and erases. SLLOCK has the same description as LLOCK. SLLOCK is not writable unless SLE is high.

12.3.2.5 Low/Mid Address Space Block Select Register (LMS)

The Low/Mid Address Space Block Select Register (LMS) provides a means to select blocks to be operated on during erase.

The LMS register is shown in Figure 12-7 and Table 12-8.

Offset: FLASH_REGS_BASE + 0x0010 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	MSEL	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	LSEL									
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 12-7. Low/Mid Address Space Block Select Register (LMS)

Table 12-8. LMS Field Descriptions

Field	Description
MSEL[1:0]	Mid Address Space Block Select. A value of 1 in the select register signifies that the block is selected for erase. A value of 0 in the select register signifies that the block is not selected. The reset value for the select registers is 0, or un-selected. The blocks must be selected (or un-selected) before doing an erase interlock write as part of the erase sequence. The select register is not writable once an interlock write is completed until MCR[DONE] is set at the completion of the requested operation, or if a high voltage operation is suspended. MSEL is also not writeable during UTest operations, when AIE is high. In the event that blocks are not present (due to configuration or total memory size), the corresponding select bits default to un-selected, and are not writable. The reset value is always 0, and register writes have no effect.
LSEL[9:0]	Low Address Space Block Select. A value of 1 in the select register signifies that the block is selected for erase. A value of 0 in the select register signifies that the block is not selected. The reset value for the select registers is 0, or un-selected. The blocks must be selected (or un-selected) before doing an erase interlock write as part of the erase sequence. The select register is not writable once an interlock write is completed until MCR[DONE] is set at the completion of the requested operation, or if a high voltage operation is suspended. LSEL is also not writeable during UTest operations, when AIE is high. In the event that blocks are not present (due to configuration or total memory size), the corresponding select bits default to un-selected, and are not writable. The reset value is always 0, and register writes have no effect.

12.3.2.6 High Address Space Block Select Register (HBS)

The High Address Space Block Select Register (HBS) provides a means to select blocks to be operated on during erase.

The HBS register is shown in Figure 12-8 and Table 12-9.

Offset: FLASH_REGS_BASE + 0x0014 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	HSEL					
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 12-8. High Address Space Block Select Register (HBS)

Table 12-9. HBS Field Descriptions

Field	Description
HSEL[5:0]	High Address Space Block Select. High Address Block Select has the same characteristics as LSEL.

12.3.2.7 Address Register (ADR)

The Address register (ADR) provides the first failing address in the event module failures (ECC or PGM/Erase state machine)

The ADR register is shown in Figure 12-9 and Table 12-10.

Offset: FLASH_REGS_BASE + 0x0018 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	SAD	0	0	0	0	0	0	0	0	0	0	ADDR				
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	ADDR													0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 12-9. Address Register (ADR)

Table 12-10. ADR Field Descriptions

Field	Description
SAD	Shadow Address. The SAD bit qualifies the address captured during an ECC Event Error, Single Bit Correction, or State Machine operation. The SAD register is not writable. 0 Address Captured is from Main Array Space. 1 Address Captured is from Shadow Array Space.
ADDR[20:3]	Address. The ADR register provides the first failing address in the event of ECC event error (MCR[EER] set), single bit correction (MCR[SBC] set), as well as providing the address of a failure that may have occurred in a state machine operation (MCR[PEG] cleared). ECC event errors take priority over single bit corrections, which take priority over state machine errors. This is especially valuable in the event of a RWW operation, where the read senses an ECC error or single bit correction, and the state machine fails simultaneously. This address is always a Double Word address that selects 64 bits. The ADR register is writable, and can be used in the UTest ECC Logic Check. If the ECC logic check is enabled (UT0[EIE] = 1) then the ADR register will not update for ECC event error, single bit correction, or state machine errors. If MCR[EER] or MCR[SBC] are set, the ADR register is locked from writing. MCR[PEG] does not affect the writability of the ADR register.

12.3.2.8 Platform Flash Configuration Register for Port *n* (PFCRP*n*)

The PFLASH configuration register for port 0 (PFCRP0) is used to specify operation of port p0 of the flash memory module. This register also has two bits (ARB and PRI) to control arbitration between the p0/p1 ports.

The PFLASH configuration register for port 1 (PFCRP1) is used to specify operation of port p1 of the flash memory module.

The PFCRP*n* register is shown in Figure 12-10 and Table 12-11.

Offset: FLASH_REGS_BASE + 0x001C (PFCRP0)
FLASH_REGS_BASE + 0x0020 (PFCRP1)

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	LBCFG				ARB	PRI	0	M8PFE	0	M6PFE	M5PFE	M4PFE	0	M2PFE	M1PFE	M0PFE
W																
Reset[1]	0	0	—[2]	—[2]	—[3]	—[3]	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	APC			WWSC		RWSC			0	DPF EN	0	IPFEN	0	PFLIM		BFEN
W																
Reset	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0

1 Reset value for PFCRP0 = 0x0800_FF00, PFCRP1 = 0x3000_FF00.

2 Reset value for port 0 is LBCFG = 0b0000, port 1 is LBCFG = 0b0011.

3 ARB and PRI are only available in PFCRP0. ARB is reset to 1 and PRI is reset to 0. For PFCRP1, ARB and PRI are both reserved, with a reset value of 0.

Figure 12-10. Platform Flash Configuration Register for Port *n* (PFCRP*n*)

Table 12-11. PFCRP0 and PFCRP1 Field Descriptions

Field	Description
LBCFG[3:0]	Line Buffer Configuration. Controls the configuration of the four line buffers in the PFLASH controller. The buffers can be organized as a pool of available resources or with a fixed partition between instruction and data buffers. In all cases, when a buffer miss occurs, it is allocated to the least recently used buffer within the group and the just-fetched entry then marked as most recently used. If the flash access is for the next sequential line, the buffer is not marked as most recently used until the given address produces a buffer hit. For PFCRP0, this field is set to 0b0000 by hardware reset. For PFCRP1, this field is set to 0b0011 by hardware reset. xx00 All four buffers are available for any flash access, i.e., there is no partitioning of the buffers based on the access type. xx01 Reserved. xx10 The buffers are partitioned into two groups: buffers 0 and 1 allocated for instruction fetches and buffers 2 and 3 for data accesses. xx11 The buffers are partitioned into two groups: buffers 0,1, 2 allocated for instruction fetches and buffer 3 for data accesses.
ARB	Arbitration Mode. This field controls which arbitration mode is used. In both fixed priority or round-robin modes, write requests are prioritized higher than read requests, and read requests are prioritized higher than speculative prefetch requests whenever both ports issue concurrent requests. This bit is set to 1 by hardware reset. 0 Fixed-priority arbitration is used; the port specified in PRI has highest fixed priority. 1 Round-robin arbitration is used. **Note:** This bit is only available in PFCRP0. For PFCRP1, treat this bit as reserved with a reset value of 0.
PRI	Fixed Priority. Controls which port has highest fixed priority when fixed priority arbitration is selected. This field has no effect when operating in round-robin mode. This bit is cleared by hardware reset. 0 Port p0 is given highest fixed priority. 1 Port p1 is given highest fixed priority. **Note:** This bit is only available in PFCRP0. For PFCRP1, treat this bit as reserved with a reset value of 0.
M*n*PFE *n* = 0:2, 4:6	Master *n* Prefetch Enable. Used to control whether prefetching may be triggered based on the AHB hmaster attribute. For example, M0PFE enables prefetching for accesses where hmaster[3:0] = 0b0000. Likewise, M4PFE enables prefetching only when hmaster[3:0] == 0b0100. Note that hmaster[3] is ignored when determining which M*n*PFE to use for a given access. These bits are cleared by hardware reset. 0 No prefetching may be triggered by this master. 1 Prefetching may be triggered by this master. **Note:** These bits refer to the master ID, not the AMBA port number, as shown in the following:

Master ID	Master Name
0	Z6 Core
1	Z0 Core
2	eDMA
3	– reserved –
4	FEC
5	MLB
6	FlexRay
7	– reserved –
8	Z6 Nexus

Table 12-11. PFCRP0 and PFCRP1 Field Descriptions (continued)

Field	Description
APC[2:0]	Address Pipelining Control. Used to control the number of cycles between pipelined access requests. This field must be set to a value corresponding to the operating frequency of the PFLASH. The settings are documented in the *PXN20 Microcontroller Data Sheet*. Higher operating frequencies require non-zero settings for this field for proper flash operation. This field is set to 0b111 by hardware reset. 000 Accesses may be pipelined back-to-back. 001 Access requests require one additional hold cycle. 010 Access requests require two additional hold cycles. ... 110 Access requests require six additional hold cycles. 111 No address pipelining. **Note:** The settings for APC and RWSC should be the same.
WWSC[1:0]	Write Wait State Control. Used to control the number of wait states to be added to the best case flash array access time for writes. This field must be set to a value corresponding to the operating frequency of the PFLASH. Higher operating frequencies require non-zero settings for this field for proper flash operation. This field is set to 0b11 by hardware reset. 00 No additional wait-states are added. 01 One additional wait-state is added. 10 Two additional wait-states are added. 11 Three additional wait-states are added.
RWSC[2:0]	Read Wait State Control. Used to control the number of wait states to be added to the best case flash array access time for reads. This field must be set to a value corresponding to the operating frequency of the PFLASH and the actual read access time of the PFLASH. This field is set to 0b111 by hardware reset. 000 No additional wait states are added. 001 One additional wait state is added. ... 111 Seven additional wait states are added. **Note:** The settings for APC and RWSC should be the same.
DPFEN	Data Prefetch Enable. Enables or disables prefetching initiated by a data read access. This field is cleared by hardware reset. 0 No prefetching is triggered by a data read access. 1 Prefetching may be triggered by any data read access.
IPFEN	Instruction Prefetch Enable. Enables or disables prefetching initiated by an instruction read access. This field is cleared by hardware reset. 0 No prefetching is triggered by an instruction read access. 1 Prefetching may be triggered by any instruction read access.

Table 12-11. PFCRP0 and PFCRP1 Field Descriptions (continued)

Field	Description
PFLIM[1:0]	PFLASH Prefetch Limit. Controls the prefetch algorithm used by the PFLASH prefetch controller. This field defines a limit on the maximum number of sequential prefetches that are attempted between buffer misses. In all situations when enabled, only a single prefetch is initiated on each buffer miss or hit. This field is cleared by hardware reset. 00 No prefetching or buffering is performed. 01 The referenced line is prefetched on a buffer miss, i.e., prefetch on miss. 1*x* the referenced line is prefetched on a buffer miss, or the next sequential line is prefetched on a buffer hit (if not already present), i.e., prefetch on miss or hit.
BFEN	PFLASH Line Read Buffers Enable. Enables or disables line read buffer hits. It is also used to invalidate the buffers. This bit is cleared by hardware reset. 0 The line read buffers are disabled from satisfying read requests, and all buffer valid bits are cleared. 1 The line read buffers are enabled to satisfy read requests on hits. Buffer valid bits may be set when the buffers are successfully filled.

12.3.2.9 Platform Flash Access Protection Register (PFAPR)

Offset: FLASH_REGS_BASE + 0x0024 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	M7AP		M6AP		M5AP		M4AP		M3AP		M2AP		M1AP		M0AP	
Reset	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	SHSACC				0	0	0	0	SHDACC				0	0	0	0
W																
Reset	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0

Figure 12-11. PFlash Access Protection Register (PFAPR)

Table 12-12. PFlash Access Protection Register (PFAPR) Field Descriptions

Field	Description
M7AP ... M0AP	Master X Access Protection. These fields are used to control whether read and write accesses to the flash are allowed based on the master ID of a requesting master. 00 No accesses may be performed by this master. 01 Only read accesses may be performed by this master. 10 Only write accesses may be performed by this master. 11 Both read and write accesses may be performed by this master.

Bit	Bus Master	Bit	Bus Master
M0AP	0 — Z6 Core	M4AP	4 — FEC
M1AP	1 — Z0 Core	M5AP	5 — MLB
M2AP	2 — eDMA	M6AP	6 — FlexRay
M3AP	3 — reserved	M7AP	7 — reserved

Table 12-12. PFlash Access Protection Register (PFAPR) Field Descriptions (continued)

Field	Description
SHSACC[7:4]	Shadow Block Supervisor Access Control. This bit field defines supervisor/user mode access control for each 4 KB sector within the shadow block region of the flash array. 0 Shadow block sector *n* can be accessed in both user and supervisor mode. 1 Shadow block sector *n* can be accessed only in supervisor mode. An attempted user mode access is terminated with an AHB error response. If the requesting bus master is the processor core, the ERROR response typically generates an instruction abort or data abort exception. This field is mapped into the shadow block (shadow_block = 0x00FF_C000) with sector base addresses of: SHSACC[4] = shadow_block + 0x0000 SHSACC[5] = shadow_block + 0x1000 SHSACC[6] = shadow_block + 0x2000 SHSACC[7] = shadow_block + 0x3000 This field is initialized by hardware reset to the value contained in address 0x3E00 of the shadow block of the flash array. An erased or unprogrammed flash sets this field to 0xFF. The contents of the PFAPR are combined with the SHSACC field to determine the final flash attributes.
SHDACC[7:4]	Shadow Block Data Access Control. This bit field defines code/data access control for each 4 KByte sector within the shadow block region of the flash array. 0 Shadow block sector *n* can only be accessed as data. An attempted instruction fetch access is terminated with an AHB error response. If the requesting bus master is the processor core, the ERROR response typically generates an instruction abort or data abort exception. 1 Shadow block sector *n* can be accessed as either code or data. This field is mapped into the shadow block using the same definition as the SHSACC field above. This field is initialized by hardware reset to the value contained in address 0x3E00 of the shadow block of the flash array. An erased or unprogrammed flash sets this field to 0xFF. The contents of the PFAPR are combined with the SHDACC field to determine the final flash attributes.

12.3.2.10 PFlash Supervisor Access Control Register (PFSACC)

Offset: FLASH_REGS_BASE + 0x0028 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	SACC[30:16]														
W																
Reset	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	SACC[15:0]															
W																
Reset	1	1	1	1	1	1	1	0	0	0	0	0	1	0	0	0

Figure 12-12. PFlash Supervisor Access Control Register (PFSACC)

Table 12-13. PFlash Supervisor Access Control Register (PFSACC) Field Descriptions

Field	Description
SACC[30:0]	Supervisor Access Control. This bit field defines supervisor/user mode access control for each sector within the main flash array. 0 Flash array sector *n* can be accessed in both user and supervisor mode. 1 Flash array sector *n* can be accessed only in supervisor mode. An attempted user mode access is terminated with an AHB error response. If the requesting bus master is the processor core, the ERROR response typically generates an instruction abort or data abort exception. The mapping of this bit field to the main flash array is defined in **Table 12-14** This field is initialized by hardware reset to the value contained in address 0x3E08 of the shadow block of the flash array. An erased or unprogrammed flash sets this field to 0xFFFF_FFFF.

Table 12-14. {S,D}ACC Register to Flash Array Mapping

Register Bit	Starting Flash Array Address	Sector Size
xACC[0]	0x00_0000	16 KB
xACC[1]	0x00_4000	16 KB
xACC[2]	0x00_8000	16 KB
xACC[3]	0x00_C000	16 KB
xACC[4]	0x01_0000	16 KB
xACC[5]	0x01_4000	16 KB
xACC[6]	0x01_8000	16 KB
xACC[7]	0x01_C000	16 KB
xACC[8]	0x02_0000	16 KB
xACC[9]	0x02_4000	16 KB
xACC[10]	0x02_8000	16 KB
xACC[11]	0x02_C000	16 KB
xACC[12]	0x03_0000	16 KB
xACC[13]	0x03_4000	16 KB
xACC[14]	0x03_8000	16 KB
xACC[15]	0x03_C000	16 KB
xACC[16]	0x04_0000	256 KB
xACC[17]	0x08_0000	256 KB
xACC[18]	0x0C_0000	256 KB
xACC[19]	0x10_0000	256 KB
xACC[20]	0x14_0000	256 KB
xACC[21]	0x18_0000	256 KB
xACC[22]	0x1C_0000	256 KB
xACC[23]	0x20_0000	256 KB
xACC[24]	0x24_0000	256 KB
xACC[25]	0x28_0000	256 KB
xACC[26]	0x2C_0000	256 KB
xACC[27]	0x30_0000	256 KB

Table 12-14. {S,D}ACC Register to Flash Array Mapping

Register Bit	Starting Flash Array Address	Sector Size
xACC[28]	0x34_0000	256 KB
xACC[29]	0x38_0000	256 KB
xACC[30]	0x3C_0000	256 KB
xACC[31]	Reserved	

12.3.2.11 PFlash Data Access Control Register (PFDACC)

Offset: FLASH_REGS_BASE + 0x002C — Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	DACC[30:16]														
W																
Reset	0	0	—	—	1	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	DACC[15:0]															
W																
Reset	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0

Figure 12-13. PFlash Data Access Control Register (PFSACC)

Table 12-15. PFlash Data Access Control Register (PFDACC) Field Descriptions

Field	Description
DACC[31:0]	Data Access Control. This bit field defines code/data access control for each sector within the main flash array. 0 Flash array sector *n* can be accessed only by a data reference. An attempted instruction fetch is terminated with an AHB error response. If the requesting bus master is the processor core, the ERROR response typically generates an instruction abort or data abort exception. 1 Flash array sector *n* can be accessed either as an instruction or data reference. The mapping of this bit field to the main flash array is defined in Table 12-14. This field is initialized by hardware reset to the value contained in address 0x3E10 of the shadow block of the flash array. An erased or unprogrammed flash sets this field to 0xFFFF_FFFF.

12.3.2.12 User Test Register 0 (UT0)

The User Test Register 0 (UT0) provides a means to control UTest. The UTest mode gives the users of the flash module the ability to perform test features on the flash. This register is only writable when the flash is put into UTest mode by writing a passcode.

Offset: FLASH_REGS_BASE + 0x003C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	UTE	SCBE	0	0	0	0	0	0	DSI							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	MRE	MRV	EIE	AIS	AIE	AID
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Figure 12-14. User Test Register 0 (UT0)

Table 12-16. UT0 Field Descriptions

Field	Description
UTE	UTest Enable. This status bit gives indication when UTest is enabled. All bits in UT0, UT1, UT2, UM0, UM1, UM2, UM3, and UM4 are locked when this bit is 0. This bit is not writeable to a 1, but may be cleared. The reset value is 0. The method to set this bit is to provide a password, and if the password matches, the UTE bit is set to reflect the status of enabled, and is enabled until it is cleared by a register write. The UTE password will only be accepted if MCR[PGM] = 0 and MCR [ERS] = 0 (program and erase are not being requested). UTE can only be cleared if UT0[AID] = 1, UT0[AIE] and UT0[EIE] = 0. While clearing UTE, writes to set AIE or set EIE will be ignored. For UTE, the password 0xF9F9_9999 must be written to the UT0 register.
SCBE	Single Bit Correction Enable. SBC enables Single Bit Correction results to be observed in MCR[SBC]. Also is used as an enable for interrupt signals created by the c90fl module (see c90fl Integration Guide). ECC corrections that occur when SBCE is cleared will not be logged. 0 Single Bit Corrections observation is disabled. 1 Single Bit Correction observation is enabled.
DSI	Data Syndrome Input. These bits enable checks of ECC logic by allowing check bits to be input into the ECC logic and then read out by doing array reads or array integrity checks. The DSI[7:0] correspond to the 8 ECC check bits on a double word.
MRE	Margin Read Enable. MRE combined with MRV enables Factory Margin Reads to be done. Margin reads are only active during Array Integrity Checks. Normal user reads are not affected by MRE. MRE is not writable if AID is low. 0 Margin reads are not enabled. 1 Margin reads are enabled during Array Integrity Checks.
MRV	Margin Read Value. MRV selects the margin level that is being checked. Margin can be checked to an erased level (MRV = 1) or to a programmed level (MRV = 0). In order for this value to be valid, MRE must also be set. MRV is not writable if AID is low. 0 Zero's margin reads are requested. 1 One's margin reads are requested.
EIE	ECC Data Input Enable. EIE enables the input registers (DSI and DAI) to be the source of data for the array. This is useful in the ECC logic check. If this bit is set, data read through a BIU read request will be from the DSI and DAI registers when an address match is achieved to the ADR register. EIE is not simultaneously writable to a 1 as UTI is being cleared to a 0. 0 Data read is from the flash array. 1 Data read is from the DSI and DAI registers.

Table 12-16. UT0 Field Descriptions (continued)

Field	Description
AIS	Array Integrity Sequence. AIS determines the address sequence to be used during array integrity checks. The default sequence (AIS = 0) is meant to replicate sequences normal “user” code follows, and thoroughly checks the read propagation paths. This sequence is proprietary. The alternative sequence (AIS = 1) is just logically sequential. It should be noted that the time to run a sequential sequence is significantly shorter than the time to run the proprietary sequence. If MRE is set, AIS has no effect. 0 Array integrity sequence is proprietary sequence. 1 Array integrity sequence is sequential.
AIE	Array Integrity Enable. AIE set to one starts the array integrity check done on all selected and unlocked blocks. The address sequence selected is determined by AIS, and the MISR (UM0 through UM4) can be checked after the operation is complete, to determine if a correct signature is obtained. Once an Array Integrity operation is requested (AIE = 1), it may be terminated by clearing AIE if the operation has finished (AID = 1) or aborted by clearing AIE if the operation is ongoing (AID = 0). AIE is not simultaneously writable to a 1 as UTI is being cleared to a 0. 0 Array integrity checks are not enabled. 1 Array integrity checks are enabled.
AID	Array Integrity Done. AID is cleared upon an Array integrity check being enabled (to signify the operation is ongoing). Once completed, AID is set to indicate that the array integrity check is complete. At this time the MISR (UMR registers) can be checked. AID can not be written, and is status only. 0 Array integrity check is ongoing. 1 Array integrity check is done.

12.3.2.13 User Test Register 1 (UT1)

The User Test Register 1 (UT1) provides added controllability to UTest.

Offset: FLASH_REGS_BASE + 0x0040 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	DAI															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	DAI															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 12-15. User Test Register 1 (UT1)

Table 12-17. UT1 Field Descriptions

Field	Description
DAI [31:0]	Data Array Input. These bits enable checks of ECC logic by allowing data bits to be input into the ECC logic and then read out by doing array reads or array integrity checks. The DAI[31:0] correspond to the 32 Array bits representing Word 0 of the double word selected in the ADR register.

12.3.2.14 User Test Register 2 (UT2)

Offset: FLASH_REGS_BASE + 0x0044 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	DAI															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	DAI															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 12-16. User Test Register 2 (UT2)

Table 12-18. UT2 Field Descriptions

Field	Description
DAI [63:32]	Data Array Input. These bits enable checks of ECC logic by allowing data bits to be input into the ECC logic and then read out by doing array reads or array integrity checks. The DAI[63:32] correspond to the 32 Array bits representing Word 1of the double word selected in the ADR register.

12.3.2.15 User Multiple Input Signature Register [0:4] (UM*n*)

The User Multiple Input Signature Registers (UM*n*) provide a means to evaluate array integrity.

Offset FLASH_REGS_BASE + 0x0048 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	MISR[31:0]																															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 12-17. User Multiple Input Signature Register 0 (UM0)

Offset FLASH_REGS_BASE + 0x004C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	MISR[63:32]																															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 12-18. User Multiple Input Signature Register 1 (UM1)

Offset FLASH_REGS_BASE + 0x0050 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	MISR[95:64]																															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 12-19. User Multiple Input Signature Register 2 (UM2)

Offset FLASH_REGS_BASE + 0x0054 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	MISR[127:96]																															
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 12-20. User Multiple Input Signature Register 3 (UM3)

Offset FLASH_REGS_BASE + 0x0058 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	MISR[143:128]																
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 12-21. User Multiple Input Signature Register 4 (UM4)

Table 12-19. UM*n* Field Descriptions

Field	Description
MISR	Multiple Input Signature Register bits. The MISR bitfields accumulate a signature from an array integrity event. The MISR captures all data fields, as well as ECC fields, and the read transfer error signal. The MISR can be seeded to any value by writing the MISR registers. The MISR register provides a means to calculate a MISR during Array Integrity operations. The MISR can be represented by the following polynomial: $x^{145} + x^6 + x^5 + x^1 + 1$ The MISR is calculated by taking the previous MISR value and then “exclusive ORing” the new data. In addition the most significant bit (in this case it is MISR[144]), is then “exclusive ORed” into input of MISR[6], MISR[5], MISR[1], and MISR[0]. The result of the “exclusive OR” is shifted left on each read. The MISR register is used in Array Integrity operations. If during address sequencing, reads extend into an invalid address location (i.e. greater than the maximum address for a given array size) or locked/un-selected blocks. Reads are still executed to the array, but the results from the array read are not deterministic. In this instance, the MISR registers is not re-calculated, and the previous value is retained.

12.4 Functional Description

12.4.1 Flash User Mode

In user mode the flash module can be read and written (register writes and interlock writes), programmed or erased. The following sub-sections define all actions that can be performed in user mode.

12.4.1.1 Flash Read and Write

The default state of the flash module is read. The main and shadow address space can be read only in the read state. The module configuration register (MCR) is always available for read. The flash module enters the read state on reset. The flash module is in the read state under four sets of conditions:

- The read state is active when the module is enabled.

- The read state is active when PGM = 1 or ERS = 1 in the MCR and high-voltage operation is ongoing (read-while-write).

NOTE

Reads done to the partition(s) being operated on (either erased or programmed) result in an error and the RWE bit in the MCR is set.

- The read state is active when PGM = 1 and PSUS = 1 in the MCR (program suspend).
- The read state is active when ERS = 1 and ESUS = 1 and PGM = 0 in the MCR (erase suspend).

NOTE

FC reads are done through the BIU. In many cases the BIU will do page buffering to allow sequential reads to be done with higher performance. This can create a data coherency issue that must be handled with software. Data coherency can be an issue after a program, erase, or shadow row operations.

In flash user mode, registers can be written. Array can be written to do interlock writes.

Reads attempted to invalid locations result in indeterminate data. Invalid locations occur when addressing is done to blocks that do not exist in non 2^n array sizes.

Interlock writes attempted to invalid locations (due to blocks that do not exist in non 2^n array sizes) will result in an interlock occurring, but attempts to program or erase these blocks will not occur since they are forced to be locked.

12.4.1.2 Read While Write (RWW)

The flash core is divided into partitions. Partitions are always comprised of two or more blocks. Partitions are used to determine read-while-write (RWW) groupings. While a write (program or erase) is being done within a given partition, a read can be simultaneously executed to any other partition. Partitions are listed in Table 12-1. Each partition in high address space comprises of two 256 KB blocks. The shadow block has unique RWW restrictions described in Section 12.4.3, Flash Shadow Block.

The FC is also divided into blocks to implement independent erase or program protection. The shadow block exists outside the normal address space and is programmed, erased, and read independently of the other blocks. The shadow block is included to support systems that require NVM for security or system initialization information.

A software mechanism is provided to independently lock or unlock each block in high-, mid-, and low-address space against program and erase. Two hardware locks are also provided to enable/disable the FC for program/erase. See Section 12.4.1.3.1, Software Locking, for more information.

12.4.1.3 Flash Programming

Programming changes the value stored in an array bit from logic 1 to logic 0 only. Programming cannot change a stored logic 0 to a logic 1. Addresses in locked/disabled blocks cannot be programmed. The user can program the values in any or all of four words within a page in a single program sequence. Word addresses are selected using bits 3:2 of the page-bound word.

Whenever a program operation occurs, ECC bits are programmed. ECC is handled on a 64-bit boundary. Thus, if only one word in any given 64-bit ECC segment is programmed, the adjoining word (in that segment) should not be programmed because ECC calculation has already completed for that 64-bit segment. Attempts to program the adjoining word will probably result in an operation failure. It is recommended that all programming operations be from 64 bits to 128 bits, and be 64-bit aligned. The programming operation should completely fill selected ECC segments within the page.

The program operation consists of the following sequence of events:

1. Change the value in the MCR[PGM] bit from a 0 to a 1.

NOTE

Ensure the block that contains the address to be programmed is unlocked. See Section 12.3.2.2, Low/Mid Address Space Block Locking Register (LML), Section 12.3.2.3, High Address Space Block Locking Register (HBL), and Section 12.3.2.4, Secondary Low/Mid Address Space Block Locking Register (SLL), for more information.

2. Write the first address to be programmed in the flash module with the program data. This write is referred to as a program data interlock write. An interlock write may be either be an aligned word or doubleword.
3. If more than one word or doubleword is to be programmed, write each additional address in the page with data to be programmed. This is referred to as a program data write. All unwritten data words default to 0xFFFF_FFFF.
4. Write a logic 1 to the MCR[EHV] bit to start the internal program sequence or skip to step 9 to terminate.
5. Wait until the MCR[DONE] bit goes high.
6. Confirm MCR[PEG] = 1.
7. Write a logic 0 to the MCR[EHV] bit.
8. If more addresses are to be programmed, return to step 2.
9. Write a logic 0 to the MCR[PGM] bit to terminate the program sequence.

The program sequence is presented graphically in Figure 12-22. The program suspend operation detailed in Figure 12-22 is discussed in Section 12.4.1.3.2, Flash Program Suspend/Resume.

The first write after a program is initiated determines the page address to be programmed. Program may be initiated with the 0 to 1 transition of the MCR[PGM] bit or by clearing the MCR[EHV] bit at the end of a previous program. This first write is referred to as an interlock write. If the program is not an erase-suspended program, the interlock write determines if the shadow or normal array space will be programmed and causes MCR[PEAS] to be set/cleared.

In the case of an erase-suspended program, the value in MCR[PEAS], is retained from the erase.

An interlock write must be performed before setting MCR[EHV]. The user may terminate a program sequence by clearing MCR[PGM] prior to setting MCR[EHV].

If multiple writes are done to the same location the data for the last write is used in programming.

While MCR[DONE] is low, MCR[EHV] is high, and MCR[PSUS] is low, the user may clear MCR[EHV], resulting in a program abort. A program abort forces the module to step 8 of the program sequence. An aborted program results in MCR[PEG] being set low, indicating a failed operation. The data space being operated on before the abort will contain indeterminate data. The user may not abort a program sequence while in program suspend.

CAUTION

Aborting a program operation leaves the flash core addresses being programmed in an indeterminate data state. This may be recovered by executing an erase on the affected blocks.

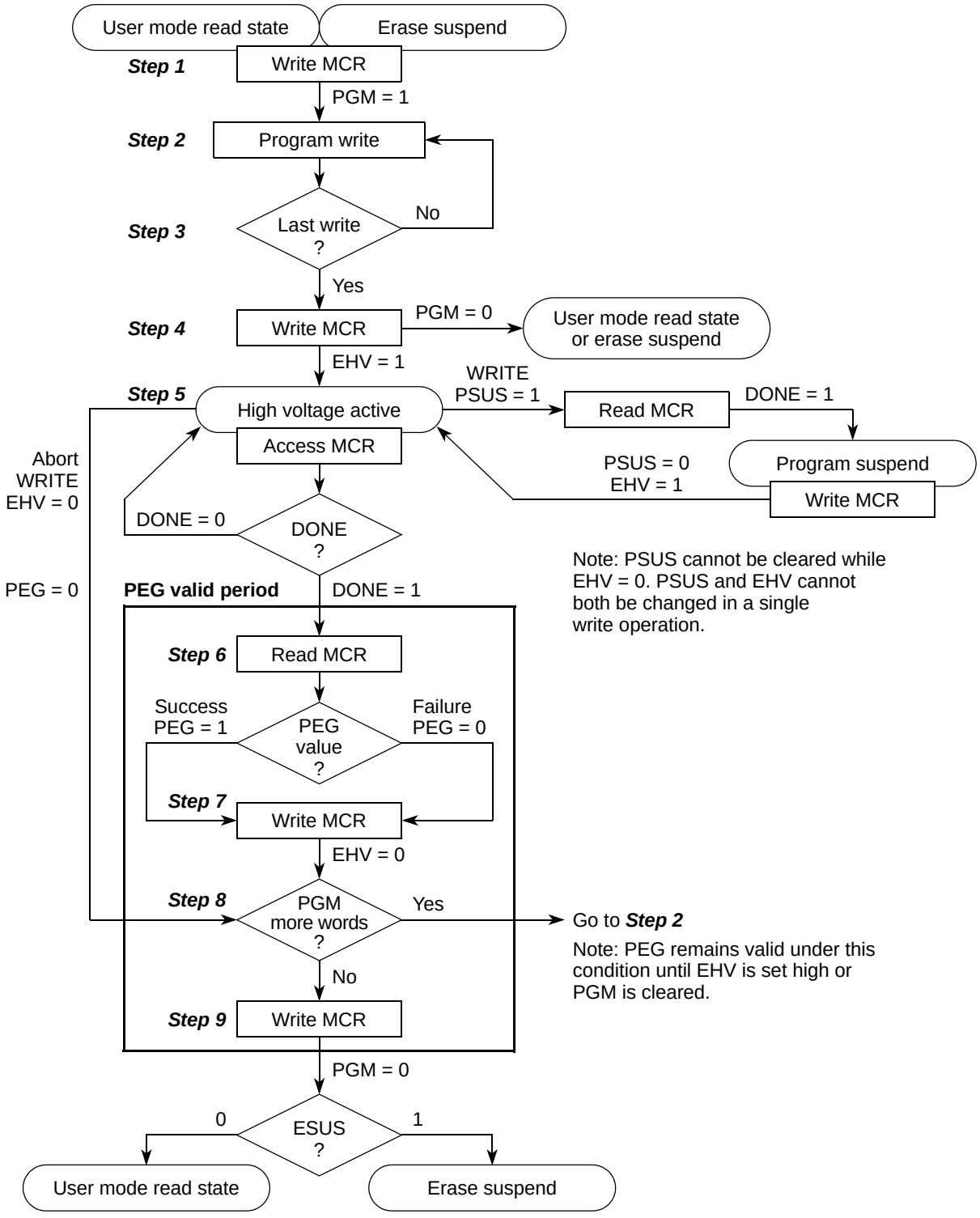


Figure 12-22. Program Sequence

12.4.1.3.1 Software Locking

A software mechanism is provided to independently lock/unlock each high-, mid-, and low-address space against program and erase.

Software locking is done through the LML (low-/mid-address space block locking register), SLL (secondary low-/mid-address space block locking register), or HBL (high-address space block locking register). These can be written through register writes and read through register reads.

When the program/erase operations are enabled through hardware, software locks are enforced through doing register writes.

12.4.1.3.2 Flash Program Suspend/Resume

The program sequence may be suspended to allow read access to the flash core. It is not possible to erase or program during a program suspend. Interlock writes should not be attempted during program suspend.

A program suspend can be initiated by changing the value of the MCR[PSUS] bit from a 0 to a 1. MCR[PSUS] can be set high at any time when MCR[PGM] and MCR[EHV] are high. A 0 to 1 transition of MCR[PSUS] causes the flash module to start the sequence to enter program suspend, which is a read state. The module is not suspended until MCR[DONE] = 1. At this time flash core reads may be attempted. After it is suspended, the flash core may be read only. Reads to the blocks being programmed/erased return indeterminate data.

The program sequence is resumed by writing a logic 0 to MCR[PSUS]. MCR[EHV] must be set to a 1 before clearing MCR[PSUS] to resume operation. When the operation resumes, the flash module continues the program sequence from one of a set of predefined points. This may extend the time required for the program operation.

12.4.1.4 Flash Erase

Erase changes the value stored in all bits of the selected blocks to logic 1. Locked or disabled blocks cannot be erased. If multiple blocks are selected for erase during an erase sequence, the blocks are erased sequentially starting with the lowest numbered block and terminating with the highest. Aborting an erase operation leaves the flash core blocks being erased in an indeterminate data state. This can be recovered by executing an erase on the affected blocks.

The erase sequence consists of the following sequence of events:

1. Change the value in the MCR[ERS] bit from 0 to a 1.
2. Select the block, or blocks, to be erased by writing 1s to the appropriate registers in LMS or HBS. If the shadow row is to be erased, this step may be skipped, and LMS and HBS are ignored. For shadow row erase, see section Section 12.4.3, Flash Shadow Block, for more information.

NOTE

Lock and select are independent. If a block is selected and locked, no erase can occur. See Section 12.3.2.2, Low/Mid Address Space Block Locking Register (LML), Section 12.3.2.3, High Address Space Block Locking Register (HBL), and Section 12.3.2.4, Secondary Low/Mid Address Space Block Locking Register (SLL), for more information.

3. Write to any address in flash. This is referred to as an erase interlock write.
4. Write a logic 1 to the MCR[EHV] bit to start an internal erase sequence or skip to step 9 to terminate.
5. Wait until the MCR[DONE] bit goes high.
6. Confirm MCR[PEG] = 1.
7. Write a logic 0 to the MCR[EHV] bit.
8. If more blocks are to be erased, return to step 2.
9. Write a logic 0 to the MCR[ERS] bit to terminate the erase.

The erase sequence is presented graphically in Figure 12-23. The erase suspend operation detailed in Figure 12-23 is discussed in section Section 12.4.1.5, Flash Erase Suspend/Resume.

After setting MCR[ERS], one write (referred to as an interlock write) must be performed before MCR[EHV] can be set to a 1. Data words written during erase sequence interlock writes are ignored. The user may terminate the erase sequence by clearing MCR[ERS] before setting MCR[EHV].

An erase operation may be aborted by clearing MCR[EHV] assuming MCR[DONE] is low, MCR[EHV] is high, and MCR[ESUS] is low. An erase abort forces the module to step 8 of the erase sequence. An aborted erase results in MCR[PEG] being set low, indicating a failed operation. The blocks being operated on before the abort contain indeterminate data. The user may not abort an erase sequence while in erase suspend.

CAUTION

Aborting an erase operation leaves the flash core blocks being erased in an indeterminate data state. This may be recovered by executing an erase on the affected blocks.

12.4.1.5 Flash Erase Suspend/Resume

The erase sequence may be suspended to allow read access to the flash core. The erase sequence may also be suspended to program (erase-suspended program) the flash core. A program started during erase suspend can be suspended. One erase suspend and one program suspend are allowed at a time during an operation. It is not possible to erase during an erase suspend, or program during a program suspend. During suspend, all reads to flash core locations targeted for program and blocks targeted for erase return indeterminate data. Programming locations in blocks targeted for erase during erase-suspended program may result in corrupted data.

An erase suspend operation is initiated by setting the MCR[ESUS] bit. MCR[ESUS] can be set to a 1 at any time when MCR[ERS] and MCR[EHV] are high and MCR[PGM] is low. A 0 to 1 transition of MCR[ESUS] causes the flash module to start the sequence which places it in erase suspend. The user must wait until MCR[DONE] = 1 before the module is suspended and further actions are attempted. After it is suspended, the array may be read or a program sequence may be initiated (erase-suspended program). Before initiating a program sequence the user must first clear MCR[EHV]. If a program sequence is initiated, the value of the MCR[PEAS] is not reset. These values are fixed at the time of the first interlock of the erase. Flash core reads from the blocks being erased while MCR[ESUS] = 1 return indeterminate data.

The erase operation is resumed by clearing the MCR[ESUS] bit. The flash continues the erase sequence from one of a set of predefined points. This can extend the time required for the erase operation.

CAUTION

In an erase-suspended program, programming flash locations in blocks which were being operated on in the erase may corrupt flash core data.

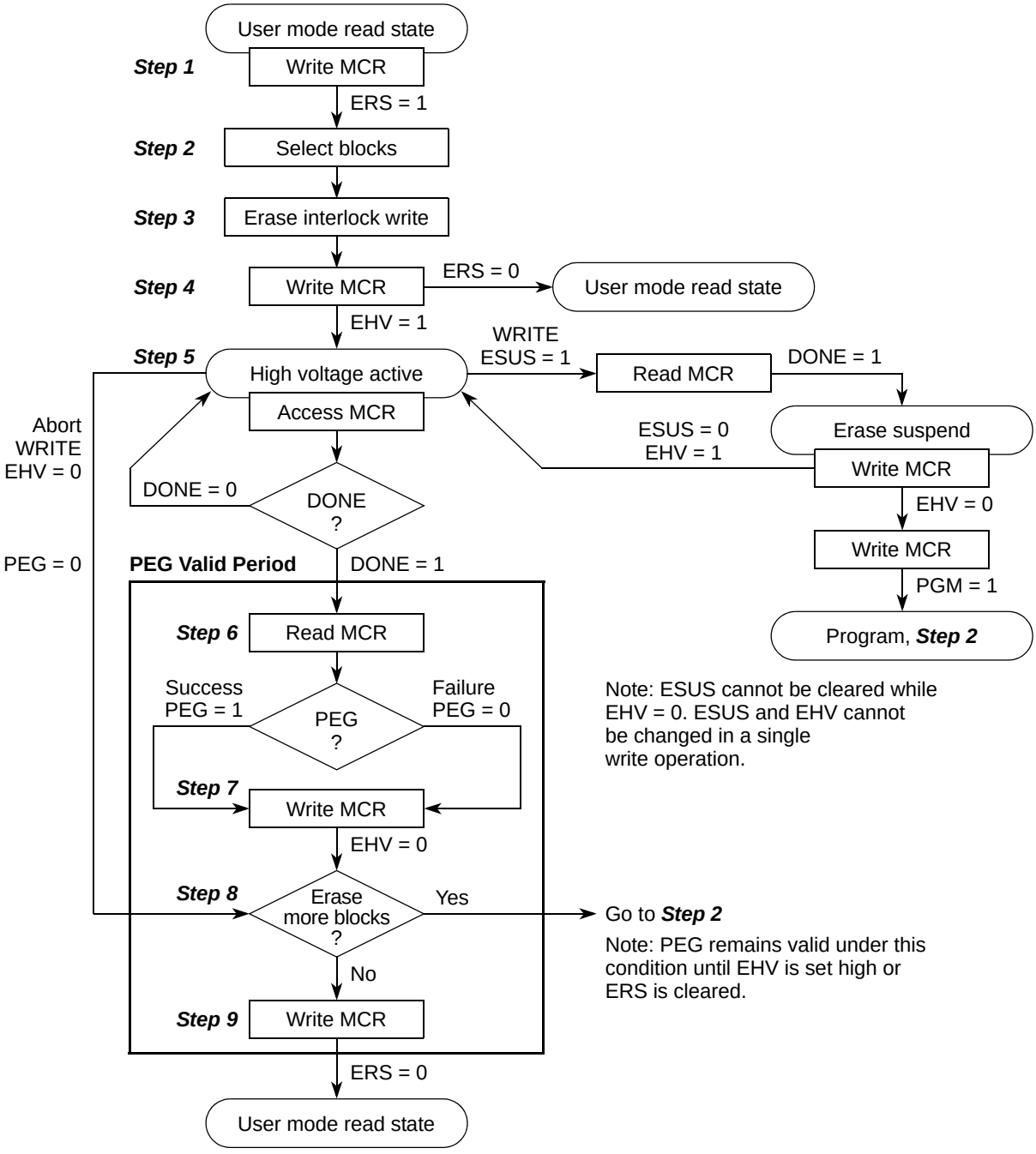


Figure 12-23. Erase Sequence

12.4.2 UTest Mode

UTest mode is a mode that customers can put the flash module in to do specific tests to check the integrity of the Flash module.

12.4.2.1 Array Integrity Self Check

Array integrity is checked using a pre-defined address sequence (based on UT0[AIS]), and this operation is executed on selected and unlocked blocks. The data to be read is customer-specific. Thus, a customer can provide user code into the flash and the correct MISR value is calculated. The customer is free to provide any random or non-random code, and a valid MISR signature is calculated. Once the operation is completed, the result of the reads can be checked by reading the MISR value to determine if an incorrect read or ECC detection was noted. Array integrity is controlled by the system clock, and it is required that the Read Wait States and Address Pipeline control registers in the BIU be set to match the user-defined frequency being used.

The array integrity check consists of the following sequence of events:

1. Enable UTest mode.
2. Select the block, or blocks to be receive array integrity check by writing ones to the appropriate registers in LMS or HBS registers.

NOTE

Locked Blocks can be tested with array integrity if selected in LMS and HBS.

NOTE

It is not possible to do UTest operations on the shadow block.

3. If desired, Set the UT0[AIS] bit to 1 for sequential addressing only.

NOTE

For normal integrity checks of the flash memory, sequential addressing is recommended. If sequential addressing is selected, BIU read requests can be done and will interrupt the array integrity sequence. Upon resuming the Array Integrity operation will continue from where it left off. The suspending and resuming of the array integrity operation is handled internally by the BIU.

NOTE

If it is required to more fully check the read path (in a diagnostic mode), it is recommend that AIS be left at 0, to use the address sequence that checks the read path more fully, and examine read transitions. This sequence takes more time. If this sequence is selected, it is recommended to not allow BIU read request interruptions, as this will diminish the effectiveness of the array integrity test, and the results will be non-deterministic.

4. Seed the MISR fields in UM0 through UM4 with the desired values.

5. Set the UT0[AIE] bit.

   If desired, the Array Integrity operation may be aborted prior to UT0[AID] going high. This may be done by clearing the UT0[AIE] bit and then continuing to the next step. It should be noted that in the event of an aborted array integrity check the MISR registers will contain a signature for the portion of the operation that was completed prior to the abort, and will not be deterministic. Prior to doing another array integrity operation, the UM0, UM1, UM2 and UM3 registers may need to be initialized to the desired seed value by doing register writes.
6. Wait until the UT0[AID] bit goes high.
7. Read values in the MISR registers (UM0 through UM4) to ensure correct signature.
8. Write a logic 0 to the UT0[AIE] bit.

12.4.2.2 Factory Margin Read

Factory margin read must be done following “Initial Factory Conditions” (see note 2 in Table A-1). One factory margin read is allowed per erase.

Factory margin read may be done to selected and unlocked blocks by combining UT0[MRE] and UT0[MRV] with the array integrity check. If UT0[MRE] is set, UT0[AIS] has no effect, and the reads will be done sequentially.

The data to be read is customer-specific. Thus, a customer can provide user code into the flash and the correct MISR value is calculated. The customer is free to provide any random or non-random code, and a valid MISR signature is calculated. Once the operation is completed, the result of the reads can be checked by reading the MISR value. Factory margin read is a self-timed event, and is independent of system clocks or wait states selected. Margin ECC corrections or detections are not done during the factory margin read test.

1. Enable UTest mode.
2. Select the block, or blocks to be receive margin read check by writing ones to the appropriate registers in LMS or HBS/EHS registers. Make sure that selected blocks are also unlocked.

   NOTE

   It is not possible to do UTest operations on the shadow block.

   NOTE

   It is possible to do User Mode array reads during the factory margin read test, if desired, but the partition rules for Read While Write used during program and erase are in effect during factory margin reads.
3. Set the UT0[MRE] bit.
4. Set the UT0[MRV] bit to the desired value to perform One’s margin or Zero’s margin.
5. Seed the MISR fields in UM0 through UM4 with the desired values.
6. Set the UT0[AIE] bit.

   If desired, the margin read operation may be aborted prior to UT0[AID] going high. This may be

done by clearing the UT0[AIE] bit and then continuing to the next step. It should be noted that in the event of an aborted margin read, the MISR registers contain a signature for the portion of the operation that was completed prior to the abort, but it is not deterministic.

7. Wait until the UT0[AID] bit goes high.
8. Read values in the MISR registers (UM0 through UM4) to ensure correct signature.
9. Write a logic 0 to the UT0[AIE] bit.

12.4.2.3 ECC Logic Check

ECC logic can be checked by providing data to be read in the UT0[DSI], UT1[DAI] and/or UT2[DAI] registers. Then array reads can be done, ensuring expected results. The ECC logic check consists of the following sequence of events:

1. Enable UTest mode.
2. Write UT0[EIE] to 1.
3. Write UT0[DSI], UT1[DAI], and/or UT2[DAI] bits to provide data and check bit values to be read. Single- or double-bit detections/corrections can be simulated by properly choosing data and check bit combinations.
4. Write double word address to receive the data input in step 3 into the ADR register.

12.4.3 Flash Shadow Block

The flash shadow block is a memory-mapped block in the flash memory map. Program and erase of the shadow block are enabled when MCR[PEAS] = 1 only. After the user has begun an erase operation on the shadow block, the operation cannot be suspended to program the main address space and vice-versa. The user must terminate the shadow erase operation to program or erase the main address space.

NOTE

If an erase of user space is requested, and a suspend is done with attempts to erase suspend program shadow space, this attempted program is directed to user space as dictated by the state of MCR[PEAS]. Likewise an attempted erase suspended program of user space, while the shadow space is being erased, is directed to shadow space as dictated by the state of MCR[PEAS].

The shadow block cannot use the RWW feature. After an operation is started in the shadow block, a read cannot be done to the shadow block, or any other block. Likewise, after an operation is started in a block in low-/mid-/high-address space, a read cannot be done in the shadow block.

The shadow block contains information about how the lock registers are reset. The first and second words can be used for reset configuration words. All other words can be used for user-defined functions or other configuration words.

The shadow block may be locked/unlocked against program or erase by using the LML or SLL discussed in Section 12.3.2, Register Descriptions.

Programming the shadow row has similar restrictions to programming the array in terms of how ECC is calculated. See Section 12.4.1.3, Flash Programming, for more information. Only one program is allowed

per 64 bit ECC segment between erases. Erase of the shadow row is done similarly as an array erase. See section Section 12.4.1.4, Flash Erase, for more information.

12.4.4 Flash Sleep Mode

Flash sleep mode is entered by setting the FDIS bit in the CPR_SOCSC register. See Section 6.2.2.12, SoC Status and Control Register (CRP_SOCSC), for more information. Once sleep mode is requested, the flash module turns off most current sources, although logic/charge pumps to enable quick recovery to read are enabled for faster wake up time than disable mode.

When in sleep mode, register access is prevented. FC accesses are also prevented until sleep mode is exited. FC reads and writes may occur as soon as sleep mode is exited.

The flash module returns to its pre-sleep state when enabled in all cases unless in the process of executing a program or erase high voltage operation at the time of sleep. If the flash module is put to sleep during a program or erase high voltage operation, the appropriate suspend bit is set to a 1. The user may resume the program or erase operation at the time the module is enabled by clearing the appropriate suspend bit. EHV must be high for the module to resume operation. If both the ESUS and PSUS bits are set to a 1 the user must clear PSUS to resume the program. The erase may be resumed after the program ends.

12.4.5 Flash Reset

A reset is the highest priority operation for the flash and terminates all other operations.

The flash uses reset to initialize register and status bits to their default reset values. If the flash is executing a program or erase operation and a reset is issued, the operation is aborted and the flash disables the high voltage logic without damage to the high-voltage circuits. Reset aborts all operations and forces the flash into user mode ready to receive accesses.

After reset is negated, register accesses can be performed, although it should be noted that registers that require updating from shadow information, or other inputs, cannot read updated until flash exits reset.

12.4.6 DMA Requests

The flash has no DMA requests.

12.4.7 Interrupt Requests

The flash has no interrupt requests.

Chapter 13
e200z6 Core (Z6)

13.1 Introduction

The core complex of the PXN20 device consists of the following:

- e200z650n3e core described in this chapter
- 32 KB unified cache memory
- 32-entry memory management unit (MMU)
- Nexus class 3 block
- Bus interface unit (BIU)

The e200z6 core is the central processing unit (CPU) in the device. The core is part of the family of CPU cores that implement versions built on the Power Architecture™ embedded category.

The core is 100% user mode compatible with the original Power PC™ user instruction set architecture (UISA). However, in the Power Architecture definition, the original floating-point resources (used by a SIMD design supporting single-precision vector and single-precision scalar operations) are provided that share the GPRs defined for integer instructions.

Throughout this book, the e200z650n3e core may also be referred to as the Z6 or the e200z6. In the context of the PXN20 device, these terms are interchangeable. Refer to the *e200z6 PowerPC™ Core Reference Manual* for more information on the e200z6 core.

The e200z0 core, used on this device as an I/O processor, is described in Chapter 14, e200z0 Core (Z0).

13.1.1 Block Diagram

Figure 13-1 shows a block diagram of the e200z6 core complex.

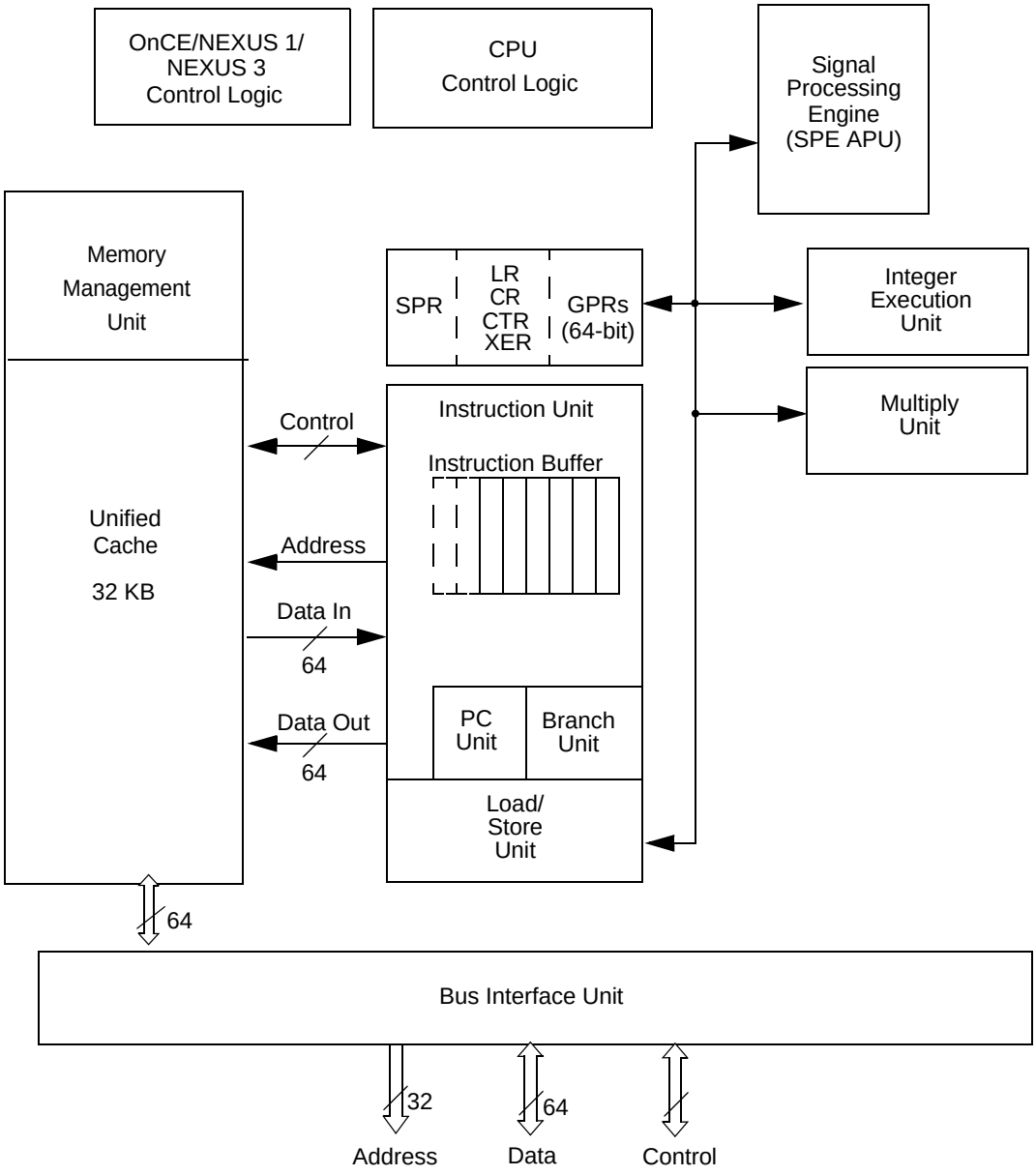


Figure 13-1. e200z6 Block Diagram

13.1.2 Overview

The e200z6 core integrates the following:

- Integer execution unit
- Branch control unit
- Instruction fetch and load/store units
- Multi-ported register file capable of sustaining three read and two write operations per clock.

Most integer instructions execute in a single-clock cycle. Branch target prefetching is performed by the branch target address cache to allow single-cycle branches in many cases.

The e200z6 core complex is built on a single-issue, 32-bit Power Architecture design with 64-bit general-purpose registers (GPRs). Power Architecture floating-point instructions are not supported in hardware, but are trapped and may be emulated by software. A signal processing extension (SPE) auxiliary processing unit (APU) is provided to support real-time fixed point and single-precision floating point operations using the general-purpose registers. All arithmetic instructions that execute in the core operate on data in the GPRs. The registers have been extended to 64-bits to support vector instructions defined by the SPE APU. These instructions operate on 16-bit or 32-bit data types, and produce vector or scalar results.

In addition to the base Power Architecture instruction set, the e200z6 core also implements the VLE (Variable Length Encoding) APU, providing improved code density.

13.1.3 Features

The following is a list of some key features of the e200z6:

- Single issue, 32-bit CPU built on the Power Architecture embedded category
- Implements the VLE APU for reduced code footprint. Refer to *EREF: A Programmer's Reference Manual for Freescale Book E Processors* and to *VLEPIM: Variable Length Encoding (VLE) Extension Programming Interface Manual*.
- In-order execution and retirement
- Precise exception handling
- Branch target address cache
  - Dedicated branch address calculation adder
  - Branch target prefetching
  - Branch lookahead buffers of depth 2
- Load/store unit: Pipelined operation supports throughput of one load or store operation per cycle
- 64-bit general-purpose register file
- Memory management unit (MMU) with 32-entry fully-associative TLB and multiple page size support
- 32 KB, 4- or 8-way set associative unified cache
- Periodic timer and watchdog functions
- Periodic system integrity can be monitored through parallel signature checks
- Signal processing extension APU supporting fixed-point and single-precision floating-point operations, using the 64-bit general-purpose register file
- Nexus class 3 real-time development unit
- Power management
  - Low-power design
  - Dynamic power management of execution units, caches, and MMUs

— Support for the wait instruction to halt synchronous activity and/or signal intent to enter low power mode to the CRP.

13.1.3.1 Instruction Unit Features

The features of the instruction unit are the following:

- 64-bit path to cache supports fetching of two 32-bit instructions per clock, or as many as four 16-bit VLE APU instructions per clock
- Instruction buffer holds as many as six sequential instructions
- Dedicated PC incrementer supporting instruction prefetches
- Branch target address cache with dedicated branch address adder, and branch lookahead logic supporting single cycle execution of successful lookahead branches

13.1.3.2 Integer Unit Features

The integer unit supports single-cycle execution of most integer instructions:

- 32-bit AU for arithmetic and comparison operations
- 32-bit LU for logical operations
- 32-bit priority encoder for count leading zeros function
- 32-bit single cycle barrel shifter for static shifts and rotates
- 32-bit mask unit for data masking and insertion
- Divider logic for signed and unsigned divides in 6–16 clocks with minimized execution timing
- Pipelined 32x32 hardware multiplier array supports 32x32->32 multiply with three clock latency, one clock throughput

13.1.3.3 Load/Store Unit Features

The load/store unit supports load, store, and the load multiple/store multiple instructions:

- 32-bit effective address adder for data memory address calculations
- Pipelined operation supports throughput of one load or store operation per cycle
- Dedicated 64-bit interface to memory supports saving and restoring as many as two registers per cycle for load multiple and store multiple word instructions

13.1.3.4 MMU Features

The features of the MMU are as follows:

- Virtual memory support
- 32-bit virtual and physical addresses
- Eight-bit process identifier
- 32-entry fully associative TLB
- Support for nine page sizes (4, 16, 64, and 256 KB; 1, 4, 16, 64, and 256 MB)
- Entry flush protection

13.1.3.5 L1 Cache Features

The features of the cache are as follows:

- 32 KB, 4- or 8-way set associative unified cache
- Copyback and writethrough support
- Eight-entry store buffer
- Push buffer
- Linefill buffer
- 32-bit address bus plus attributes and control
- Separate unidirectional 64-bit read data bus and 64-bit write data bus
- Supports cache line locking
- Supports way allocation
- Cache power usage can be minimized

13.1.3.6 BIU Features

The features of the e200z6 BIU are as follows:

- 32-bit address bus plus attributes and control
- Separate unidirectional 64-bit read data bus and 64-bit write data bus
- Overlapped, in-order accesses

13.1.4 Microarchitecture Summary

The e200z6 processor utilizes a seven stage pipeline for instruction execution. The instruction fetch 1, instruction fetch 2, instruction decode/register file read, execute1, execute2/memory access1, execute3/memory access2, and register writeback stages operate in an overlapped fashion, allowing single clock instruction execution for most instructions.

The integer execution unit consists of a 32-bit arithmetic unit (AU), a logic unit (LU), a 32-bit barrel shifter (shifter), a mask-insertion unit (MIU), a condition register manipulation unit (CRU), a count-leading-zeros unit (CLZ), a 32 x 32 hardware multiplier array, result feed-forward hardware, and support hardware for division.

Most arithmetic and logical operations are executed in a single cycle with the exception of multiply, which is implemented with a pipelined hardware array, and the divide instructions. A count-leading-zeros unit operates in a single clock cycle.

The instruction unit contains a PC incrementer and a dedicated branch address adder to minimize delays during change of flow operations. Sequential prefetching is performed to ensure a supply of instructions into the execution pipeline. Branch target prefetching is performed to accelerate taken branches. Prefetched instructions are placed into an instruction buffer capable of holding six sequential instructions.

Branch target addresses are calculated in parallel with branch instruction decode, resulting in execution time of three clocks. Conditional branches which are not taken execute in a single clock. Branches with successful lookahead and target prefetching have an effective execution time of one clock.

Memory load and store operations are provided for byte, halfword, word (32-bit), and doubleword data with automatic zero or sign extension of byte and halfword load data as well as optional byte reversal of data. These instructions can be pipelined to allow effective single cycle throughput. Load and store multiple word instructions allow low overhead context save and restore operations. The load/store unit contains a dedicated effective address adder to allow effective address generation to be optimized.

The condition register unit supports the condition register (CR) and condition register operations defined by the Power Architecture embedded category. The condition register consists of eight 4-bit fields that reflect the results of certain operations, such as move, integer and floating-point compare, arithmetic, and logical instructions, and provide a mechanism for testing and branching.

Vectored and auto-vectored interrupts are supported by the CPU. Vectored interrupt support is provided to allow multiple interrupt sources to have unique interrupt handlers invoked with no software overhead.

The SPE APU supports vector instructions operating on 16- and 32-bit fixed-point data types, as well as 32-bit IEEE® 754 single-precision floating-point formats, and supports single-precision floating-point operations in a pipelined fashion. The 64-bit general-purpose register file is used for source and destination operands, and there is a unified storage model for single-precision floating-point data types of 32-bits and the normal integer type. Low latency fixed-point and floating-point add, subtract, multiply, divide, compare, and conversion operations are provided, and most operations can be pipelined.

13.2 Core Registers and Programmer's Model

This section describes the registers implemented in the e200z6 core. It includes an overview of registers defined by the Power Architecture embedded category, highlighting differences in how these registers are implemented in the e200z6 core, and provides a detailed description of core-specific registers. Full descriptions of the architecture-defined register set are provided in the Power Architecture embedded category.

The Power Architecture embedded category defines register-to-register operations for all computational instructions. Source data for these instructions are accessed from the on-chip registers or are provided as immediate values embedded in the opcode. The three-register instruction format allows specification of a target register distinct from the two source registers, thus preserving the original data for use by other instructions. Data is transferred between memory and registers with explicit load and store instructions only.

e200z6 extends the general-purpose registers to 64-bits for supporting SPE APU operations. Power Architecture instructions operate on the lower 32 bits of the GPRs only, and the upper 32 bits are unaffected by these instructions. SPE vector instructions operate on the entire 64-bit register. The SPE APU defines load and store instructions for transferring 64-bit values to/from memory.

Figure 13-2 and Figure 13-3 show the complete e200z6 register set. Figure 13-2 shows the registers that are accessible while in supervisor mode, and Figure 13-3 shows the set of registers that are accessible while in user mode. The number to the right of the special-purpose registers (SPRs) is the decimal number used in the instruction syntax to access the register (for example, the integer exception register (XER) is SPR 1).

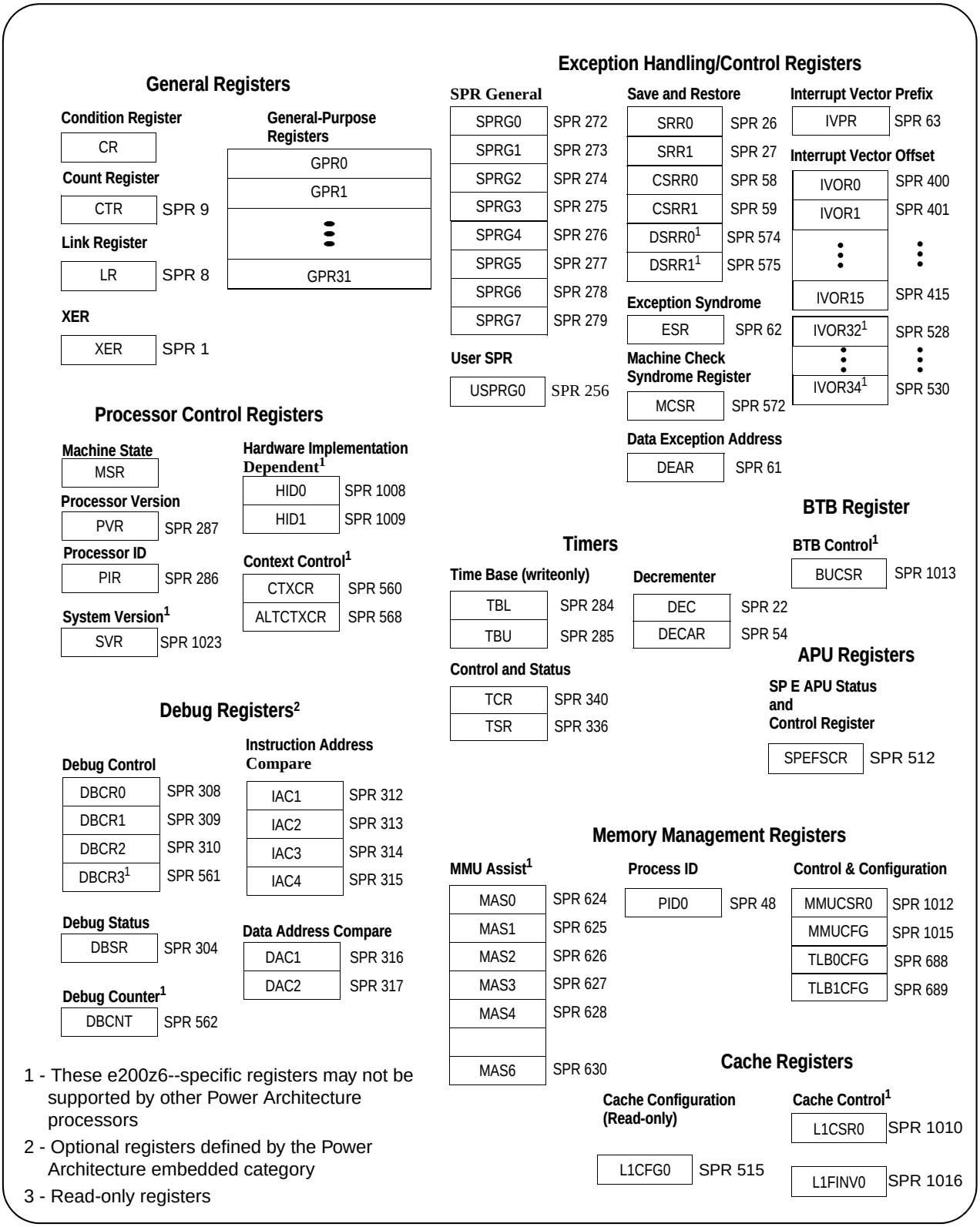


Figure 13-2. Supervisor Mode Programmer's Model

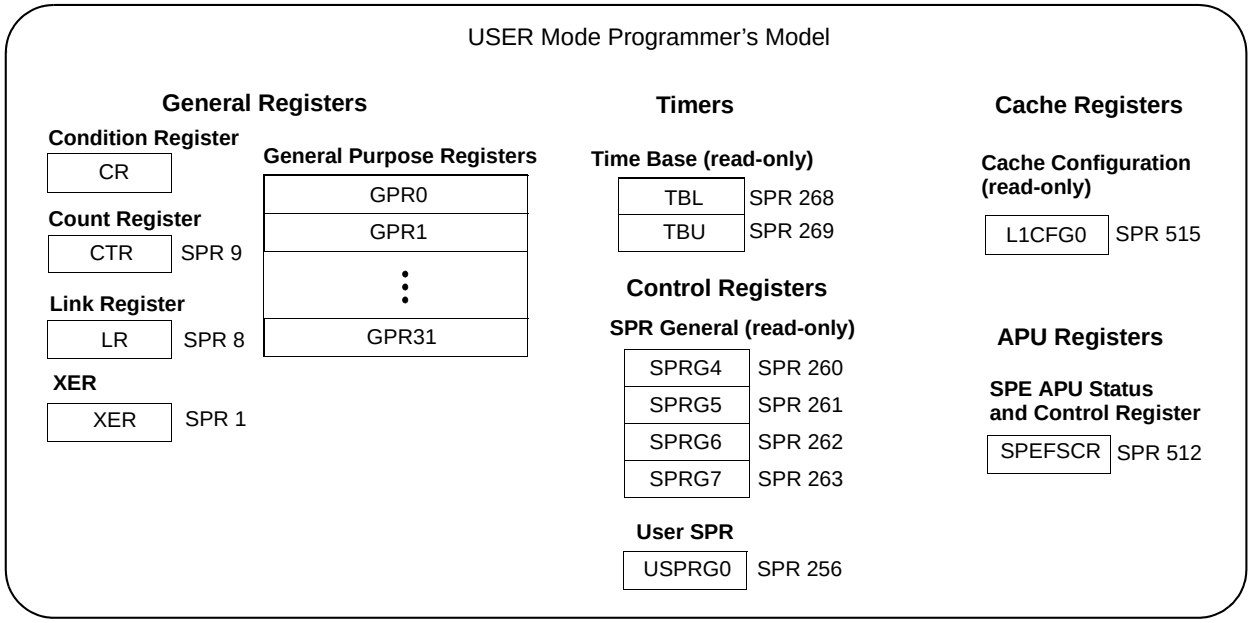

USER Mode Programmer's Model
General Registers
Condition Register
CR
Count Register
CTR
SPR 9
Link Register
LR
SPR 8
XER
XER
SPR 1
General Purpose Registers
GPR0
GPR1
GPR31
Timers
Time Base (read-only)
TBL
SPR 268
TBU
SPR 269
Control Registers
SPR General (read-only)
SPRG4
SPR 260
SPRG5
SPR 261
SPRG6
SPR 262
SPRG7
SPR 263
User SPR
USPRG0
SPR 256
Cache Registers
Cache Configuration (read-only)
L1CFG0
SPR 515
APU Registers
SPE APU Status and Control Register
SPEFSCR
SPR 512

Figure 13-3. User Mode Programmer's Model

13.2.1 Power Architecture Registers

The e200z6 core supports most of the registers defined by the Power Architecture embedded category. Notable exceptions are the floating point registers FPR0–FPR31 and FPSCR. The e200z6 does not support the Power Architecture floating point architecture in hardware. The supported Power Architecture embedded category registers are described as follows:

13.2.1.1 User-Level Registers

The user-level registers can be accessed by all software with user or supervisor privileges. They include the following:

- General-purpose registers (GPRs). The thirty-two 64-bit GPRs (GPR0–GPR31) serve as data source or destination registers for integer and SPE APU instructions and provide data for generating addresses. Power Architecture Book E instructions affect only the lower 32 bits of the GPRs.
  SPE APU instructions are provided which operate on the entire 64-bit register.
- Condition register (CR). The 32-bit CR consists of eight 4-bit fields, CR0–CR7, that reflect results of certain arithmetic operations and provide a mechanism for testing and branching.
- The remaining user-level registers are SPRs. Note that the Power Architecture provides the **mtspr** and **mfspr** instructions for accessing SPRs.
- Integer exception register (XER). The XER indicates overflow and carries for integer operations.
- Link register (LR). The LR provides the branch target address for the branch conditional to link register (**bclr**, **bclrl**) instructions, and is used to hold the address of the instruction that follows a branch and link instruction, typically used for linking to subroutines.
- Count register (CTR). The CTR holds a loop count that can be decremented during execution of appropriately coded branch instructions. The CTR also provides the branch target address for the branch conditional to count register (**bcctr**, **bcctrl**) instructions.
- The time-base facility (TB) consists of two 32-bit registers: time-base upper (TBU) and time-base lower (TBL). These two registers are accessible in a read-only fashion to user-level software.
- SPRG–SPRG7. The Power Architecture Book E architecture defines software-use special purpose registers (SPRGs). SPRG4–SPRG7 are accessible as read-only by user-level software. The e200z6 does not allow user mode access to the SPRG3 register (defined as implementation dependent by Book E).
- USPRG0. The Power Architecture Book E architecture defines user software-use special purpose register USPRG0 which is accessible in a read-write fashion by user-level software.

13.2.1.2 Supervisor-Level Only Registers

In addition to the registers accessible in user mode, supervisor-level software has access to additional control and status registers an operating system used for configuration, exception handling, and other operating system functions. The Power Architecture embedded category defines the following supervisor-level registers:

- Processor control registers
  - Machine state register (MSR). The MSR defines the state of the processor. The MSR can be modified by the move to machine state register (**mtmsr**), system call (**sc)**, and return from exception (**rfi, rfci, rfdi)** instructions. It can be read by the move from machine state register (**mfmsr)** instruction. When an interrupt occurs, the contents of the MSR are saved to one of the machine state save/restore registers (SRR1, CSRR1, DSRR1).
  - Processor version register (PVR). This register is a read-only register that identifies the version (model) and revision level of the processor built on the Power Architecture.

Table 13-1. PVR Values, and Processor Type and Version Numbers

Device	Core	PVR Value	Type	Version
PXN20	e200z6	0x8112_0000	0x11	0x2

  - Processor identification register (PIR). This read-only register is provided to distinguish the processor from other processors in the system.
- Storage control register
  - Process ID register (PID, also referred to as PID0). This register is provided to indicate the current process or task identifier. It is used by the MMU as an extension to the effective address, and by external Nexus 2/3/4 modules for ownership trace message generation. The Power Architecture embedded category allows for multiple PIDs; e200z6 implements only one.
- Interrupt registers
  - Data exception address register (DEAR). After a data storage interrupt (DSI), alignment interrupt, or data TLB miss interrupt, the DEAR is set to the effective address (EA) generated by the faulting instruction.
  - Software-use special purpose registers (SPRGs). The SPRG0–SPRG7 registers are provided for operating system use.
  - Exception syndrome register (ESR). The ESR register provides a syndrome to differentiate between the different kinds of exceptions which can generate the same interrupt.
  - Interrupt vector prefix register (IVPR) and the interrupt vector offset registers (IVOR1–IVOR15). These registers together provide the address of the interrupt handler for different classes of interrupts.
  - Save/restore registers (SRR0, SRR1). SRR0 holds the effective address for the instruction at which execution resumes when an **rfi** instruction is executed at the end of a non-critical class interrupt handler routine. SRR1 is used to save machine state on a non-critical interrupt, and stores the MSR register contents. The MSR value is restored when an **rfi** instruction is executed at the end of a non-critical class interrupt handler routine.
  - Critical save/restore registers (CSRR0, CSRR1). CSRR0 holds the effective address for the instruction at which execution resumes when an **rfci** instruction is executed at the end of a critical class interrupt handler routine. CSRR1 is used to save machine state on a critical interrupt, and stores the MSR register contents. The MSR value is restored when an **rfci** instruction is executed at the end of a critical class interrupt routine.
- Debug facility registers

- Debug control registers (DBCR0–DBCR2). These registers provide control for enabling and configuring debug events.
- Debug status register (DBSR). This register contains debug event status.
- Instruction address compare registers (IAC1–IAC4). These registers contain addresses and/or masks which are used to specify instruction address compare debug events.
- Data address compare registers (DAC1, DAC2). These registers contain addresses and/or masks which are used to specify data address compare debug events.
- e200z6 does not implement the data value compare registers (DVC1, DVC2).

- Timer registers
  - The clock inputs for the timers are connected to the internal system clock.
  - Time base (TB). The TB is a 64-bit structure provided for maintaining the time of day and operating interval timers. The TB consists of two 32-bit registers, time-base upper (TBU) and time-base lower (TBL). The time-base registers can be written to by supervisor-level software only, but can be read by both user and supervisor-level software.
  - Decrementer register (DEC). This register is a 32-bit decrementing counter that provides a mechanism for causing a decrementer exception after a programmable delay.
  - Decrementer auto-reload (DECAR). This register is provided to support the auto-reload feature of the decrementer.
  - Timer control register (TCR). This register controls decrementer, fixed-interval timer, and watchdog timer options.
  - Timer status register (TSR). This register contains status on timer events and the most recent watchdog timer-initiated processor reset.

For more details about these registers, refer to the Power Architecture embedded category specifications.

13.2.2 Core-Specific Registers

The Power Architecture embedded category allows implementation-specific registers. Implementation-specific registers incorporated in the e200z6 core are described in this section.

13.2.2.1 User-Level Registers

The user-level registers can be accessed by all software with either user or supervisor privileges. They include the following:

- Signal processing extension APU status and control register (SPEFSCR). The SPEFSCR contains all fixed-point and floating-point exception signal bits, exception summary bits, exception enable bits, and rounding control bits needed for compliance with the IEEE 754 standard.
- The L1 cache configuration register (L1CFG0). This read-only register allows software to query the configuration of the L1 unified cache.

13.2.2.2 Supervisor-Level Registers

The following supervisor-level registers are defined in the e200z6 core in addition to the Power Architecture embedded category registers described previously:

- Configuration registers
  - Hardware implementation-dependent 0 (HID0) controls processor and system functions.
  - Hardware implementation-dependent 1 (HID1) controls processor and system functions.
- Exception handling and control registers
  - Debug save and restore registers (DSRR0, DSRR1). DSRR0 holds the effective address for the instruction at which execution resumes when an **rfdi** instruction is executed at the end of a debug interrupt handler routine. DSRR1 is used to save machine state on a debug interrupt, and stores the MSR register contents. The MSR value is restored when an **rfdi** instruction is executed at the end of a debug interrupt handler routine.
  - When enabled, the DSRR0 register is used to save the address of the instruction at which execution continues when **rfdi** executes at the end of a debug interrupt handler routine.
  - Interrupt vector offset registers (IVOR32–IVOR34). These registers provide the address of the interrupt handler for different classes of interrupts.
- Debug facility registers
  - Debug control register 3 (DBCR3) controls for debug functions not described in the Power Architecture embedded category.
  - Debug counter register (DBCNT) provides counter capability for debug functions.
- Cache registers
  - L1 cache configuration register (L1CFG0) is a read-only register that allows software to query the configuration of the L1 cache.
  - L1 cache control and status register (L1CSR0) controls the operation of the L1 unified cache such as cache enabling, cache invalidation, cache locking, or 8 etc.
  - L1 cache flush and invalidate register (L1FINV0) controls software flushing and invalidation of the L1 unified cache.
- Memory management unit registers
  - MMU configuration register (MMUCFG) is a read-only register that allows software to query the configuration of the MMU.
  - MMU assist (MAS0–MAS4, MAS6) registers provide the interface to the core from the memory management unit.
  - MMU control and status register (MMUCSR0) controls invalidation of the MMU.
  - TLB configuration registers (TLBCFG0, TLBCFG1) are read-only registers that allow software to query the configuration of the TLBs.
- System version register (SVR) is a read-only and identifies the version (model) and revision level of the system with an e200z6 processor built on the Power Architecture embedded category.

For more details about these registers, refer to the e200z6 core reference documentation.

13.2.3 e200z6 Core Complex Features Not Supported in the Device

The device implements a subset of the e200z6 core complex features. The e200z6 core complex features that are *not* supported in the device are described in Table 13-2.

Table 13-2. e200z6 Features Not Supported in the Device Core

Function / Category	Description
Disabled events	The unconditional debug event (UDE) is not supported.
Power management	e200z6 core halted state and stopped state are not supported.
Power management	The following low-power modes are not supported: Doze mode Nap mode Sleep mode Time-base interrupt wake-up from low-power mode is not supported.
Power management	Core wake up is not supported. MSR[WE] bit in the machine state register is not supported. The OCR[WKUP] bit in the e200z6 OnCE control register (OCR) has no effect.
Machine check	The machine check input pin is not supported. HID0 [EMCP] has no effect, and MCSR[MCP] always reads a negated value.
PVR value	Least significant halfword of processor version register (PVR) is 0x 0000, that contains the following bitfields: MBG Use = 0x00 MBG Rev = 0x0 MBG ID = 0x0 The PVR register has two bitfields in the device.
Reservation management	Reservation management logic external to the e200z6 is not implemented.
Verification	The system version register (SVR) of the e200z6 is 0x 0000_0000.
Time base	The decrement counters are always enabled in the e200z6.
Time Base	The timer external clock is not connected to a clock; Do not select the timer external clock.
Context control	The CTXCR and ALTCXTCR registers are not supported.

13.3 Functional Description

The following sections describe the functions of the e200z6 core blocks.

13.3.1 Memory Management Unit (MMU)

The memory management unit (MMU) is an implementation built on the Power Architecture embedded category with a 32-entry fully associative translation lookaside buffer (TLB). The Power Architecture embedded category divides the effective and real address space into pages. The page represents the granularity of effective address translation, permission control, and memory/cache attributes. The e200z6 MMU supports the following nine page sizes: (4, 16, 64, and 256 KB, 1, 4, 16, 64, and 256 MB).

13.3.1.1 Translation Lookaside Buffer (TLB)

The TLB consists of a 32-entry, fully associative content addressable memory (CAM) array. To perform a lookup, the CAM is searched in parallel for a matching TLB entry. The contents of this TLB entry are then concatenated with the page offset of the original effective address. The result constitutes the physical address of the access. Table 13-3 shows the TLB entry bit definitions.

Table 13-3. TLB Entry Bit Definitions

Field	Comments
V	Valid bit for entry
TS	Translation address space (compared against AS bit)
TID[0:7]	Translation ID (compared against PID0 or ‘0’)
EPN[0:19]	Effective page number (compared against effective address)
RPN[0:19]	Real page number (translated address)
SIZE[0:3]	Page size = 4 KB,16 KB, 64 KB, 256 KB, 1 MB, 4 MB, 16 MB, 64 MB, 256 MB
SX, SW, SR	Supervisor execute, write, and read permission bits
UX, UW, UR	User execute, write, and read permission bits
WIMGE	Translation attributes (Write-through required, cache-inhibited, memory coherence required, guarded, endian)
U0–U3	User bits—used by software only
IPROT	Invalidation protect
VLE	VLE page indicator

The TLB is accessed indirectly through several MMU assist (MAS) registers. Software can read and write to the MMU assist registers with **mtspr** (move to SPR) and **mfspr** (move from SPR) instructions. The MMU registers contain information related to reading and writing an entry in the TLB. Data is read from the TLB into the MAS registers with a **tlbre** (TLB read entry) instruction. Data is written to the TLB from the MAS registers with a **tlbwe** (TLB write entry) instruction.

Refer to Section 13.3.1.5, MMU Assist Registers (MAS[0:4], MAS[6]), and the *e200z6 PowerPC™ Core Reference Manual* for more details.

13.3.1.2 Translation Flow

The effective address, concatenated with the address space value of the MSR bit (MSR[IS] or MSR[DS]), is compared to the number of bits of the EPN field and the TS field of TLB entries. If the contents of the effective address plus the address space bit matches the EPN field and TS bit of the TLB entry, that TLB entry is a candidate for a possible translation match. In addition to a match in the EPN field and TS, a matching TLB entry must match with the current process ID of the access (in PID0), or have a TID value of 0, indicating the entry is globally shared among all processes.

Figure 13-4 shows the translation match logic for the effective address plus its attributes, collectively called the virtual address, and how it is compared with the corresponding fields in the TLB entries.

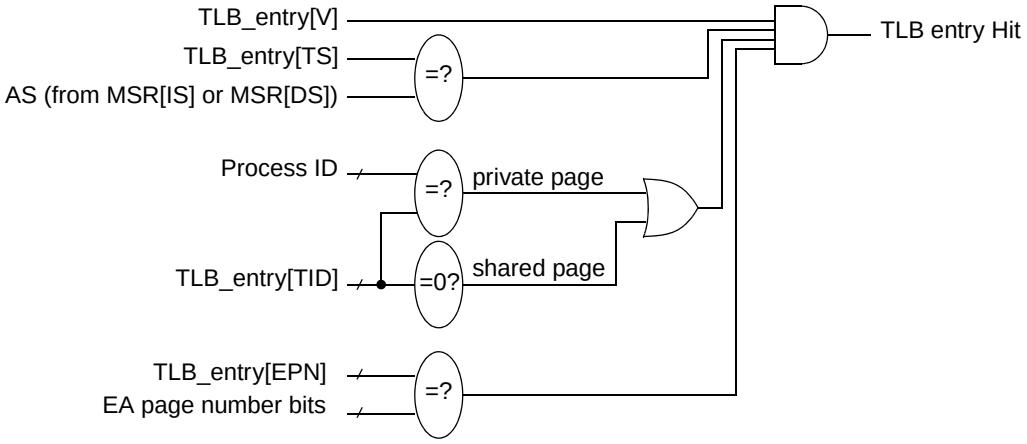


Figure 13-4. Virtual Address and TLB-Entry Compare Process

13.3.1.3 Effective to Real Address Translation

Instruction accesses are generated by sequential instruction fetches or due to a change in program flow (branches and interrupts). Data accesses are generated by load, store, and cache management instructions. The instruction fetch, branch, and load/store units generate 32-bit effective addresses. The MMU translates this effective address to a 32-bit real address which is then used for memory accesses. Figure 13-5 shows the effective to real address translation flow.

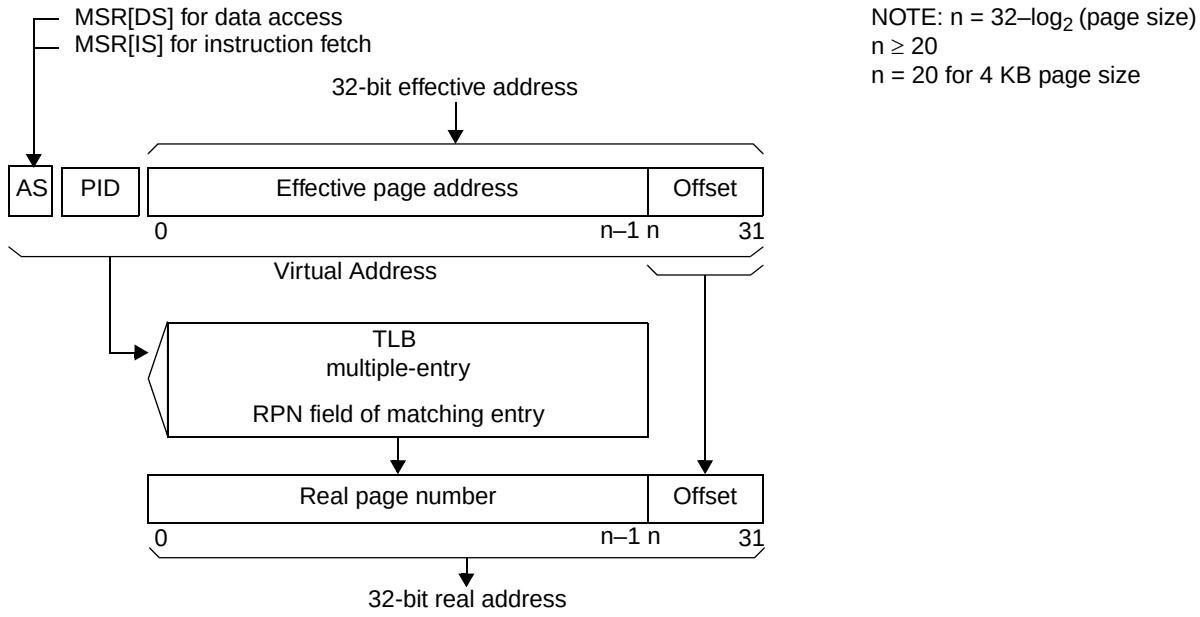


Figure 13-5. Effective to Real Address Translation Flow

13.3.1.4 Permissions

The application software can restrict access to virtual pages by selectively granting permissions for user mode read, write, and execute, and supervisor mode read, write, and execute on a per-page basis. For

example, program code might be execute-only and data structures can be mapped as read/write/no-execute.

The UX, SX, UW, SW, UR, and SR access control bits support selective permissions for access control:

- SR—Supervisor read permission. Allows loads and load-type cache management instructions to access the page while in supervisor mode.
- SW—Supervisor write permission. Allows stores and store-type cache management instructions to access the page while in supervisor mode.
- SX—Supervisor execute permission. Allows instruction fetches to access the page and instructions to be executed from the page while in supervisor mode.
- UR—User read permission. Allows loads and load-type cache management instructions to access the page while in user mode.
- UW—User write permission. Allows stores and store-type cache management instructions to access the page while in user mode.
- UX—User execute permission. Allows instruction fetches to access the page and instructions to be executed from the page while in user mode.

If the translation match was successful, the permission bits are checked as shown in Figure 13-6. If the access is not allowed by the access permission mechanism, the processor generates an instruction or data storage interrupt (ISI or DSI).

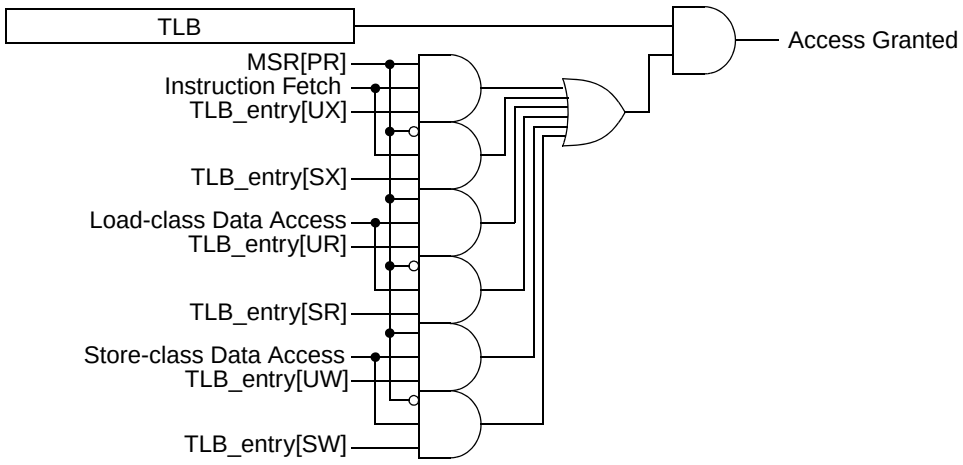


Figure 13-6. Granting of Access Permission

13.3.1.5 MMU Assist Registers (MAS[0:4], MAS[6])

The e200z6 uses six special purpose registers (MAS[0], MAS[1], MAS[2], MAS[3], MAS[4], and MAS[6]) to facilitate reading, writing, and searching the TLBs. The MAS registers can be read or written using the **mfspr** and **mtspr** instructions. The e200z6 does not implement the MAS5 register, present in other Freescale EIS designs, because the **tlbsx** instruction only searches based on a single SPID value.

For more information on the MAS*n* registers is available in the *e200z6 PowerPC*^{TM} *Core Reference Manual*.

13.3.1.5.1 MAS[0] Register

The MAS[0] register is shown in Figure 13-7.

SPR: 624 Access: Read/write

	0 1	2 3	4 5 6 7	8 9 10 11	12 13 14 15	16 17 18 19	20 21 22 23	24 25 26 27	28 29 30 31
R W	—	TLBSEL	—		ESEL	—			NV
Reset	Undefined on Power Up — Unchanged on Reset								

Figure 13-7. MAS Register 0 Format—MAS[0]

MAS[0] fields are defined in Table 13-4.

Table 13-4. MAS[0]—MMU Read/Write and Replacement Control

Field	Description
TLBSEL	Selects TLB for access. 01 TLB1 (ignored by the e200z6, write to 01 for future compatibility)
ESEL	Entry select for TLB1.
NV	Next replacement victim for TLB1 (software managed). Software updates this field; it is copied to the ESEL field on a TLB error.

13.3.1.5.2 MAS[1] Register

The MAS[1] register is shown in Figure 13-8.

SPR: 625 Access: Read/write

	0	1	2 3	4 5 6 7	8 9 10 11	12 13 14 15	16 17 18	19	20 21 22 23	24 25 26 27	28 29 30 31
R W	VALID	IPROT	—		TD		—	TS	TSIZE	—	
Reset	Undefined on Power Up — Unchanged on Reset										

Figure 13-8. MMU Assist Register 1—MAS[1]

MAS[1] fields are defined in Table 13-5.

Table 13-5. MAS[1]—Descriptor Context and Configuration Control

Field	Description
VALID	TLB entry valid. 0 This TLB entry is invalid. 1 This TLB entry is valid.
IPROT	Invalidation protect 0 Entry is not protected from invalidation. 1 Entry is protected from invalidation. Protects TLB entry from invalidation by **tlbivax** (TLB1 only), or flash invalidates through MMUCSR0[TLB1_FI].
TID	Translation ID bits. This field is compared with the current process IDs of the effective address to be translated. A TID value of 0 defines an entry as global and matches with all process IDs.

Table 13-5. MAS[1]—Descriptor Context and Configuration Control (continued)

Field	Description
TS	Translation address space. This bit is compared with the IS or DS fields of the MSR (depending on the type of access) to determine if this TLB entry can be used for translation.
TSIZE	Entry page size. Supported page sizes are: 0001 4 KB 0110 4 MB 0010 16 KB 0111 16 MB 0011 64 KB 1000 64 MB 0100 256 KB 1001 256 MB 0101 1 MB All other values are undefined.

13.3.1.5.3 MAS[2] Register

The MAS[2] register is shown in Figure 13-9.

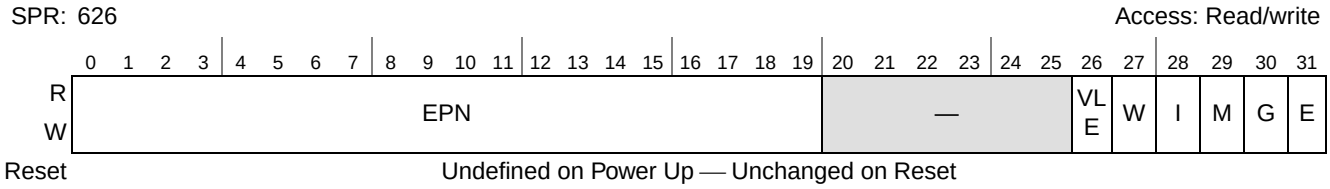


Figure 13-9. MMU Assist Register 2—MAS[2]

MAS[2] fields are defined in Table 13-6.

Table 13-6. MAS[2]—EPN and Page Attributes

Field	Description
EPN	Effective page number [0:19].
VLE	Power Architecture VLE. 0 This page is a standard Book E page. 1 This page is a Power Architecture VLE page.
W	Write-through required. 0 This page is considered write-back with respect to the caches in the system. 1 All stores performed to this page are written through to main memory.
I	Cache inhibited. 0 This page is considered cacheable. 1 This page is considered cache-inhibited.
M	Memory coherence required.The e200z6 does not support the memory coherence required attribute, and thus it is ignored. 0 Memory coherence is not required. 1 Memory coherence is required.

Table 13-6. MAS[2]—EPN and Page Attributes (continued)

Field	Description
G	Guarded. The e200z6 ignores the guarded attribute because no speculative or out-of-order processing is performed. 0 Access to this page are not guarded, and can be performed before it is known if they are required by the sequential execution model. 1 All loads and stores to this page are performed without speculation (that is, they are known to be required).
E	Endianness. Determines endianness for the corresponding page. 0 The page is accessed in big-endian byte order. 1 The page is accessed in true little-endian byte order.

13.3.1.5.4 MAS[3] Register

The MAS[3] register is shown in Figure 13-10.

SPR: 627 Access: Read/write

	Permission Bits																													
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	RPN																				—		U0	U1	U2	U3	UX	SX	UW	SW	UR	SR
Reset	Undefined on Power Up — Unchanged on Reset																															

Figure 13-10. MMU Assist Register 3—MAS[3]

MAS[3] fields are defined in Table 13-7.

Table 13-7. MAS[3]—RPN and Access Control

Field	Description
RPN	Real page number. Only bits that correspond to a page number are valid. Bits that represent offsets within a page are ignored and must be zero.
U0–U3	User bits.
PERMIS	Permission bits (UX, SX, UW, SW, UR, SR).

13.3.1.5.5 MAS[4] Register

The MAS[4] register is shown in Figure 13-11.

SPR: 628 Access: Read/write

	Default WIMGE values																													
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	—		TLBSELD		—										TIDSELD		—				TSIZED				—		VL ED	WD	ID	MD	GD	ED
Reset	Undefined on Power Up — Unchanged on Reset																															

Figure 13-11. MMU Assist Register 4 MAS[4]

MAS[4] fields are defined in Table 13-8.

Table 13-8. MAS[4]—Hardware Replacement Assist Configuration Register

Field	Description
TLBSELD	Default TLB selected 01 TLB1 (ignored by the e200z6, write as 01 for future compatibility)
TIDSELD	Default PID# to load TID from 00 PID0 01 Reserved, do not use 10 Reserved, do not use 11 TIDZ (0x00)) (Use all zeros, the globally shared value)
TSIZED	Default TSIZE value
VLED	Default VLED value
DWIMGE	Default WIMGE values

13.3.1.5.6 MAS[6] Register

The MAS[6] register is shown in Figure 13-12.

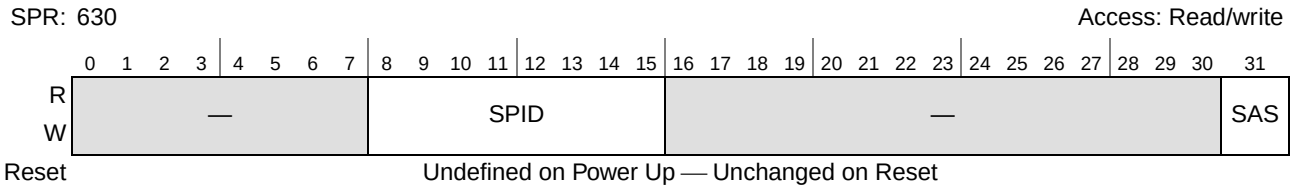


Figure 13-12. MMU Assist Register 6—MAS[6]

MAS[6] fields are defined in Table 13-9.

Table 13-9. MAS[6]—TLB Search Context Register 0

Field	Description
SPID	PID value for searches
SAS	AS value for searches

13.3.2 L1 Cache

The e200z6 processor supports a 32 KB, 4- or 8-way set-associative, unified (instruction and data) cache with a 32-byte line size. The cache improves system performance by providing low-latency data to the e200z6 instruction and data pipelines, which decouples processor performance from system memory performance. The cache is virtually indexed and physically tagged. The e200z6 does not provide hardware support for cache coherency in a multi-master environment. Software must be used to maintain cache coherency with other possible bus masters.

Both instruction and data accesses are performed using a single bus connected to the cache. Addresses from the processor to the cache are virtual addresses used to index the cache array. The MMU provides the virtual to physical translation for use in performing the cache tag compare. If the physical address matches a valid cache tag entry, the access hits in the cache. For a read operation, the cache supplies the data to the processor, and for a write operation, the data from the processor updates the cache. If the access does not

match a valid cache tag entry (misses in the cache) or a write access must be written through to memory, the cache performs a bus cycle on the system bus. Figure 13-13 shows a block diagram of the unified cache in the e200z6.

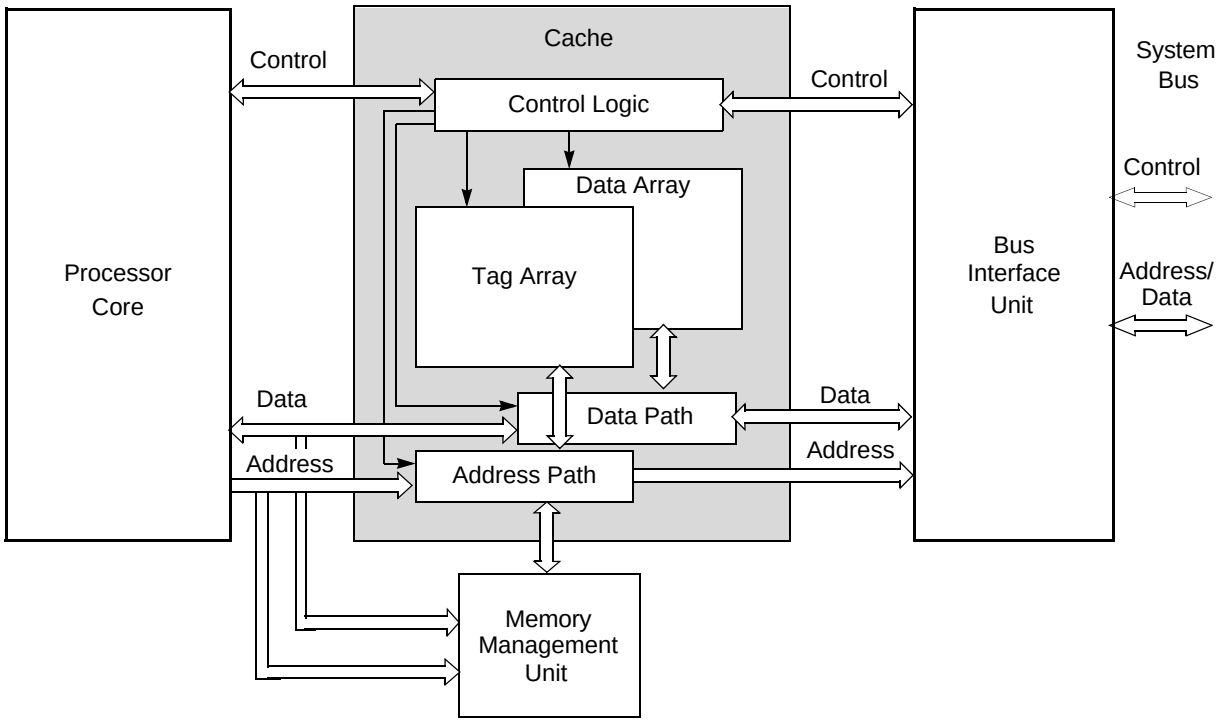


Figure 13-13. e200z6 Unified Cache Block Diagram

13.3.2.1 Cache Organization

The e200z6 cache is organized as 4 or 8 ways of 128 sets with each line containing 32 bytes (four doublewords) plus parity of storage. Figure 13-14 illustrates the cache organization, terminology used, the cache line format, and cache tag formats.

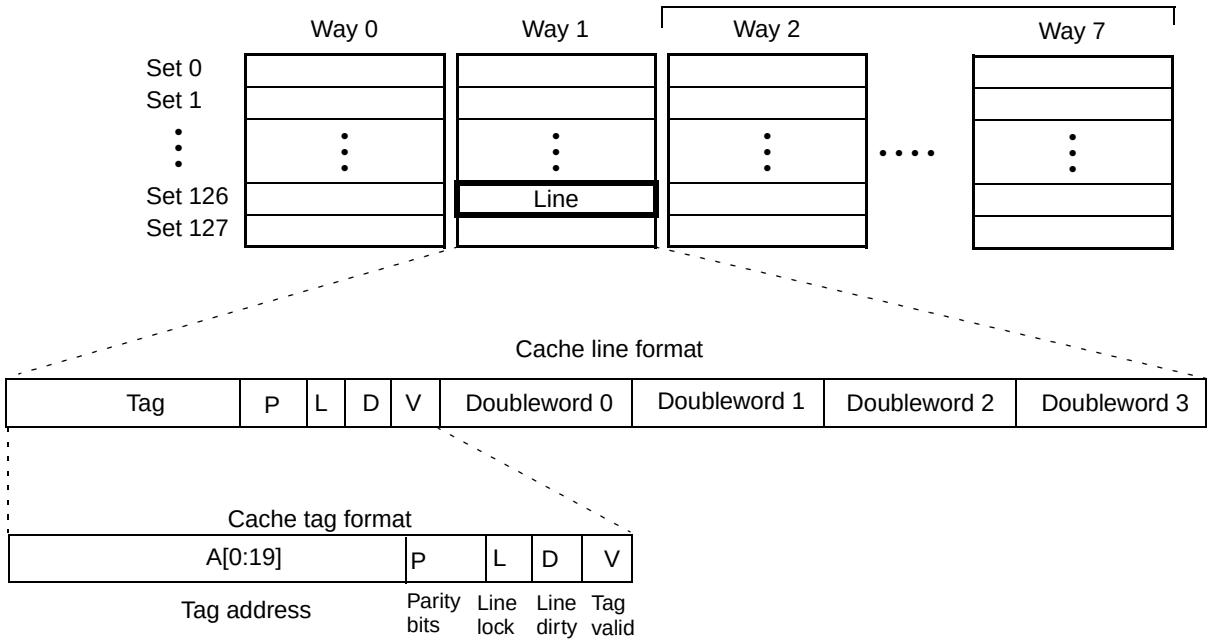


Figure 13-14. Cache Organization and Line Format

13.3.2.2 Cache Lookup

After it is enabled, the unified cache is searched for a tag match on all instruction fetches and data accesses from the CPU. If a match is found, the cached data is forwarded on a read access to the instruction fetch unit or the load/store unit (data access), or is updated on a write access, and can also be written-through to memory if required.

When a read miss occurs, if there is a TLB hit and the cache inhibit bit (WIMGE = 0bx0xxx) of the hitting TLB entry is clear, the translated physical address is used to fetch a four doubleword cache line beginning with the requested doubleword (critical doubleword first). The line is fetched and placed into the appropriate cache block and the critical doubleword is forwarded to the CPU. Subsequent doublewords can be streamed to the CPU if they have been requested and streaming is enabled via the DSTRM bit in the L1CSR0 register.

During a cache line fill, doublewords received from the bus are placed into a cache linefill buffer, and can be forwarded (streamed) to the CPU if such a request is pending. Accesses from the CPU following delivery of the critical doubleword can be satisfied from the cache (hit under fill, non-blocking) or from the linefill buffer if the requested information has been already received.

The cache always fills an entire line, thereby providing validity on a line-by-line basis. A cache line is always in one of the following states: invalid, valid, or dirty (and valid). For invalid lines, the V bit is clear, causing the cache line to be ignored during lookups. Valid lines have their V bit set and D bit cleared, indicating the line contains valid data consistent with memory. Dirty cache lines have the D and V bits set, indicating that the line has valid entries that have not been written to memory. In addition, a cache line can be locked (L bit set) indicating the line is not available for replacement.

The cache must be invalidated after a hardware reset; a hardware reset does not invalidate the cache lines. Following initial power-up, the cache contents are undefined. If the L, D, or V bits are set on any lines, the software must invalidate cache before the cache is enabled.

Figure 13-15 illustrates the general flow of cache operation.

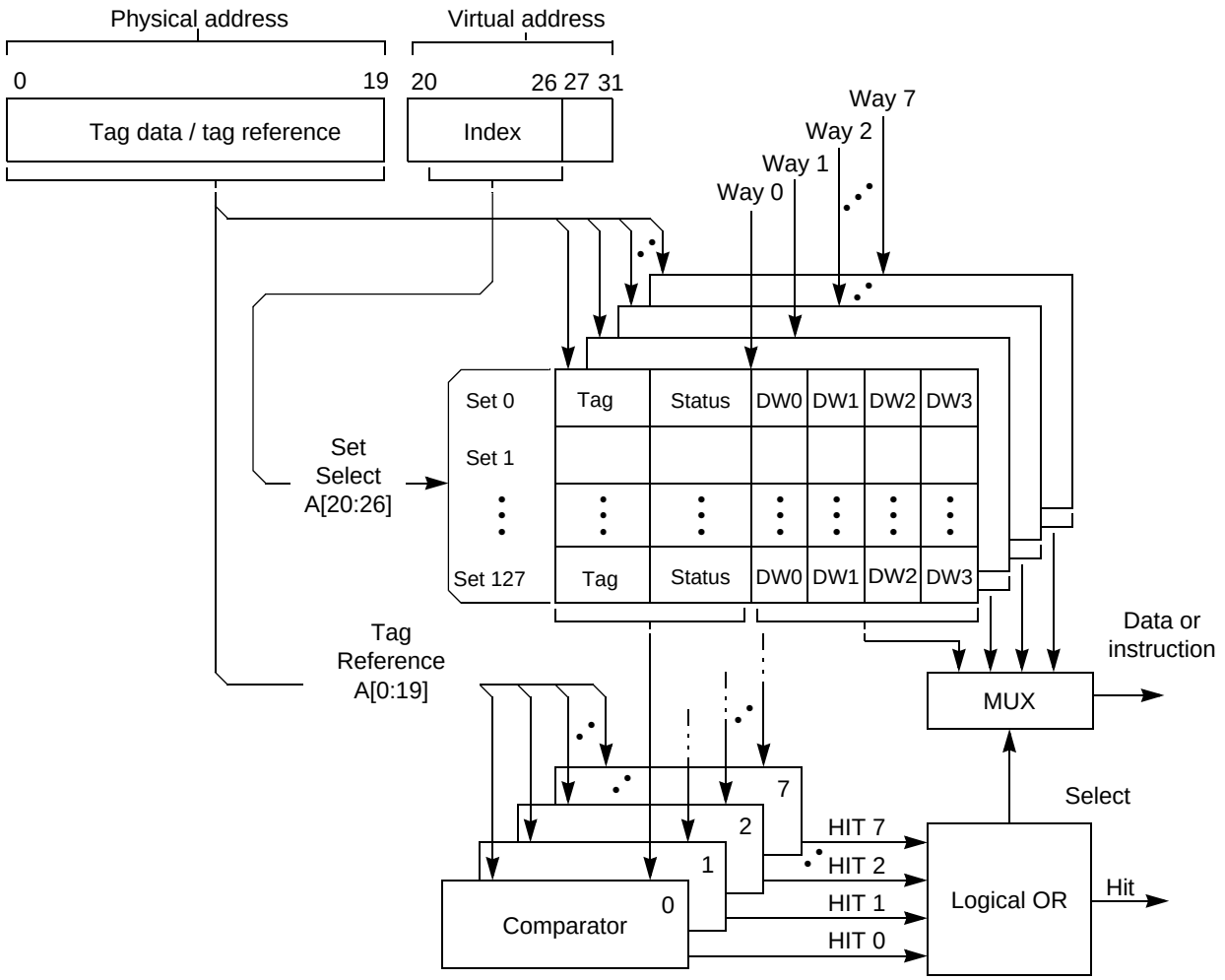


Figure 13-15. Cache Lookup Flow

To determine if the address is already allocated in the cache the following steps are taken:

1. The cache set index, virtual address bits A[20:26] are used to select one cache set. A set is defined as the grouping of four or eight lines (one from each way), corresponding to the same index into the cache array.
2. The higher order physical address bits A[0:19] are used as a tag reference or used to update the cache line tag field.
3. The four or eight tags from the selected cache set are compared with the tag reference. If any one of the tags matches the tag reference and the tag status is valid, a cache hit has occurred.

4. Virtual address bits A[27:28] are used to select one of the four doublewords in each line. A cache hit indicates that the selected doubleword in that cache line contains valid data (for a read access), or can be written with new data depending on the status of the W access control bit from the MMU (for a write access).

13.3.2.3 Cache Line Replacement Algorithm

On a cache read miss, the cache controller uses a pseudo-round-robin replacement algorithm to determine which cache line is selected to be replaced. There is a single replacement counter for the entire cache. The replacement algorithm acts as follows: on a miss, if the replacement pointer is pointing to a way which is not enabled for replacement by the type of the miss access (the selected line or way is locked), it is incremented until an available way is selected (if any). After a cache line is successfully filled without error, the replacement pointer increments to point to the next cache way.

13.3.2.4 Cache Power Reduction

The device provides additional user control over cache power utilization via the L1CSR0[WID], [AWID], [WDD], and [AWDD] way disable bits and the L1CSR0[WAM] control bit. When WAM is set to 1, ways that are disabled for allocation on miss by a particular access type (instruction or data) via the L1CSR0[WID], [AWID], [WDD], and [AWDD] way disable bits are also disabled (not selected) during normal cache lookup operations, thus avoiding the power associated with reading tag and data information for a disabled way. This provides the capability of disabling some ways for instruction accesses and some ways for data accesses to reduce power. In doing so however, certain restrictions must be followed, and the ability to lock by way is no longer functional, since a locked way would never be accessed.

When setting WAM to 1, restrictions are required to avoid coherency issues between instruction and data accesses, and to avoid multiple ways hitting on a given access. The restriction on coherency is due to the fact that a given line could possibly be present twice in the cache; a copy in a way disabled for instruction access which can be read and written by data accesses, and a second copy in a way disabled for data access which can be executed via an instruction fetch. A data write to the line results in the possibility of instruction fetches obtaining stale data, in the same manner as exists in a non-unified cache. Another restriction is that multiple hits to the same line must be avoided on any given instruction or data access. This must be avoided by controlling the ways via the L1CSR0[WID,] [WDD], [AWID], and [AWDD] bits such that no common way exists that can be accessed by both instructions and data, or by ensuring that MMU permissions are set so that no cacheable page has X (execute) permission which also has R (read) or W (write) permission, i.e.,can be cacheable and accessed with both instruction and data accesses.

When WAM is set to 1, ways disabled for instruction access are not affected by the **icbt**, **icblc**, **icbtls**, and **icbi** instructions. Ways disabled for data accesses are not affected by the **dcba**, **dcbf**, **dcbi**, **dcblc**, **dcbst**, **dcbt**, **dcbtls**, **dcbtst**, **dcbtstls**, and **dcbz** instructions. Cache control operations using L1CSR0[CINV] and L1FINV0 operations are not affected by the WAM setting and proceed normally.

13.3.2.5 L1 Cache Control and Status Register 0 (L1CSR0)

The L1 cache control and status register 0 (L1CSR0) is a 32-bit register. The L1CSR0 register is accessed using a **mfspr** or **mtspr** instruction. The SPR number for L1CSR0 is 1010 in decimal. The L1CSR0 register is shown in Figure 13-16.

The correct sequence necessary to change the value of LSCSR0 is as follows:

1. **msync**
2. **isync**
3. **mtspr** L1CSR0

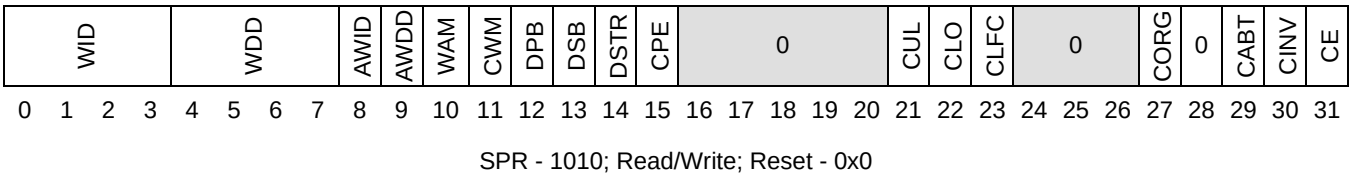


Figure 13-16. L1 Cache Control and Status Register 0 (L1CSR0)

The L1CSR0 bits are described in Table 13-10.

Table 13-10. L1CSR0 Field Descriptions

Bits	Name	Description
0:3	WID	Way instruction disable. 0 The corresponding way is available for replacement by instruction miss line fills. 1 The corresponding way is not available for replacement by instruction miss line fills. Bit 0 corresponds to way 0. Bit 1 corresponds to way 1. Bit 2 corresponds to way 2. Bit 3 corresponds to way 3. The WID and WDD bits can be used for locking ways of the cache, and also are used in determining the replacement policy of the cache.
4:7	WDD	Way data disable. 0 The corresponding way is available for replacement by data miss line fills. 1 The corresponding way is not available for replacement by data miss line fills. Bit 4 corresponds to way 0. Bit 5 corresponds to way 1. Bit 6 corresponds to way 2. Bit 7 corresponds to way 3. The WID and WDD bits can be used for locking ways of the cache, and also are used in determining the replacement policy of the cache.
8	AWID	Additional ways instruction disable. 0 Additional ways beyond 0–3 are available for replacement by instruction miss line fills. 1 Additional ways beyond 0–3 are not available for replacement by instruction miss line fills. For the 32KB 8-way cache, ways 4–7 are considered additional ways. When configured as a 4-way cache, this bit is ignored.

Table 13-10. L1CSR0 Field Descriptions (continued)

Bits	Name	Description
9	AWDD	Additional ways data disable. 0 Additional ways beyond 0–3 are available for replacement by data miss line fills. 1 Additional ways beyond 0–3 are not available for replacement by data miss line fills. For the 32KB 8-way cache, ways 4–7 are considered additional ways. When configured as a 4-way cache, this bit is ignored.
10	WAM	Way access mode 0 Disable way access is checked not enabled for replacement on an access type are still checked for a cache hit for accesses of that type but are not replaced by an access miss of that type. 1 Ways not enabled for replacement on a particular access type (instruction vs. data) via the AWID, WID, AWDD, and WDD fields are disabled and no lookup is performed for accesses of that type. Selecting WAM = 1 helps minimize power consumption. Software *must* ensure that the instruction to data coherency is maintained when using the power-saving feature of the WAM control. Cache must be invalidated prior to changing the value of this bit. Use of a dcbf followed by an icbi, msync, isync for modified lines which can be executed is required to maintain proper operation.
11	CWM	Cache write mode 0 Cache operates in writethrough mode 1 Cache operates in copyback mode When set to writethrough mode, the “W” page attribute from an optional MMU is ignored and all writes are treated as writethrough required. When set, write accesses are performed in copyback mode unless the “W” page attribute from an optional MMU is set.
12	DPB	Disable push buffer 0 Push buffer enabled 1 Push buffer disabled
13	DSB	Disable store buffer 0 store buffer enabled 1 store buffer disabled
14	DSTRM	Disable streaming 0 streaming is enabled 1 streaming is disabled
15	CPE	Cache parity enable 0 parity checking is disabled 1 parity checking is enabled
16–20	—	Reserved
21	CUL	Cache unable to lock. Indicates a lock set instruction was not effective in locking a cache line. This bit is set by hardware on an “unable to lock” condition (other than lock overflows), and remains set until cleared by software writing 0 to this bit location.
22	CLO	Cache lock overflow Indicates a lock overflow (overlocking) condition occurred. This bit is set by hardware on an “overlocking” condition, and remains set until cleared by software writing 0 to this bit location.
23	CLFC	Cache lock bits flash clear. When written to a 1, a cache lock bits flash clear operation is initiated by hardware. After this is complete, this bit is reset to 0. Writing a 1 while a flash clear operation is in progress results in an undefined operation. Writing a 0 to this bit while a flash clear operation is in progress is ignored. Cache lock bits flash clear operations require approximately cycles to complete. Clearing occurs regardless of the enable (CE) value.

Table 13-10. L1CSR0 Field Descriptions (continued)

Bits	Name	Description
24–26	—	Reserved
27	CORG	Cache organization 0 The cache is organized as 128 sets and 8 ways 1 The cache is organized as 256 sets and 4 ways. Selecting CORG = 1 helps minimize power consumption.
28	—	Reserved
29	CABT	Cache operation aborted. Indicates a cache invalidate or a cache lock bits flash clear operation was aborted prior to completion. This bit is set by hardware on an aborted condition, and remains set until cleared by software writing 0 to this bit location.
30	CINV	Cache invalidate 0 No cache invalidate 1 Cache invalidation operation When written to a 1, a cache invalidation operation is initiated by hardware. After this is complete, this bit is reset to 0. Writing a 1 while an invalidation operation is in progress results in an undefined operation. Writing a 0 to this bit while an invalidation operation is in progress is ignored. Cache invalidation operations require approximately cycles to complete. Invalidation occurs regardless of the enable (CE) value.
31	CE	Cache Enable 0 Cache is disabled 1 Cache is enabled. When disabled, cache lookups are not performed for normal load or store accesses. Other L1CSR0 cache control operations are still available. Also, operation of the store buffer is not affected by CE.

13.3.2.6 L1 Cache Configuration Register 0 (L1CFG0)

The L1 cache configuration register 0 (L1CFG0) is a 32-bit read-only register. L1CFG0 provides information about the configuration of the Zen Z650n3e L1 cache design. The contents of the L1CFG0 register can be read using a **mfspr** instruction. The SPR number for L1CFG0 is 515 in decimal. The L1CFG0 register is shown in Figure 13-17.

CARCH	CWPA	CFAHA	CFISWA	0	CBSIZE	CREPL	CLA	CPA	CNWAY	CSIZE
0 1	2	3	4	5 6	7 8	9 10	11	12	13 14 15 16 17 18 19 20	21 22 23 24 25 26 27 28 29 30 31
01	1	0	1	0 0	00	10	1	1	00000111 (8 way) / 00000011 (4 way)	00000100000 (32 KB)

SPR - 515; Read-only

Figure 13-17. L1 Cache Configuration Register 0 (L1CFG0)

The L1CFG0 bits are described in Table 13-11.

Table 13-11. L1CFG0 Field Descriptions

Bits	Name	Description
0–1	CARCH	Cache architecture 01 The cache architecture is unified
2	CWPA	Cache way partitioning available 1 The cache supports partitioning of way availability for I/D accesses
3	CFAHA	Cache flush all by hardware available 0 The cache does not support flush all in hardware
4	CFISWA	Cache flush/invalidate by set and way available 1 The cache supports flushing/invalidation by set and way via the L1FINV0 spr
5–6	—	Reserved—read as zeros
7–8	CBSIZE	Cache block size 00 The cache implements a block size of 32 bytes
9–10	CREPL	Cache replacement policy 10 The cache implements a pseudo-round-robin replacement policy
11	CLA	Cache locking APU available 1 The cache implements the line locking APU
12	CPA	Cache parity available 1 The cache implements parity
13:20	CNWAY	Number of ways in the data cache 0x03 - The cache is 4-way set-associative 0x07 - The cache is 8-way set-associative
21:31	CSIZE	Cache size 0x020 - The size of the cache is 32 KB

13.3.3 Interrupt Types

The interrupts implemented in the device and the exception conditions that cause them are listed in Table 13-12.

Table 13-12. Interrupts and Conditions

Interrupt Type	Interrupt Vector Offset Register	Enables[1]	Core Register in Which State Information is Saved	Causing Conditions
System reset	none, vector to 0xFFFF_FFFC			• Reset by assertion of $\overline{\text{RESET}}$ • Watchdog timer reset control • Debug reset control
Critical input	IVOR0[2]	CE = 1		• Non-maskable interrupt request

Table 13-12. Interrupts and Conditions (continued)

Interrupt Type	Interrupt Vector Offset Register	Enables[1]	Core Register in Which State Information is Saved	Causing Conditions
Machine check	IVOR 1	ME	CSSR[0:1]	• Machine check exception and MSR[ME] = 1 • ISI, ITLB error on first instruction fetch for an exception handler • Parity error signaled on cache access • Write bus error on buffered store or cache line push
Data storage	IVOR 2	—	SRR[0:1]	• Access control • Byte ordering due to misaligned access across page boundary to pages with mismatched E bits • Cache locking exception • Precise external termination error
Instruction storage	IVOR 3	—	SRR[0:1]	• Access control. • Precise external termination error.
External input	IVOR 4[2]	EE, src	SRR[0:1]	External interrupt is asserted and MSR[EE] = 1
Alignment	IVOR 5	—	SRR[0:1]	• **lmw**, **stmw** not word aligned • **lwarx** or **stwcx.** not word aligned • **dcbz** with disabled cache or no cache present, or to W or I storage • SPE ld and st instructions not properly aligned
Program	IVOR 6	—	SRR[0:1]	Illegal, privileged, trap, FP enabled, AP enabled, unimplemented operation
Floating-point unavailable	IVOR 7	—	SRR[0:1]	MSR[FP] = 0 and attempt to execute a Book E floating point operation
System call	IVOR 8	—	SRR[0:1]	Execution of the system call (**sc**) instruction
AP unavailable	IVOR 9	—	SRR[0:1]	Unused by e200z6
Decrementer	IVOR 10	EE, DIE	SRR[0:1]	Decrementer timeout, and as specified in *Book E: Enhanced PowerPC™ Architecture, Rev 1.0*, Ch. 8, pg. 194–195 and in the *e200z6 PowerPC™ Core Reference Manual,* Rev 0.
Fixed interval timer	IVOR 11	EE, FIE	SRR[0:1]	Fixed-interval timer timeout and as specified in *Book E: Enhanced PowerPC™ Architecture, Rev 1.0*, Ch. 8, pg. 195–196 and in the *e200z6 PowerPC™ Core Reference Manual,* Rev 0.
Watchdog timer	IVOR 12	CE, WIE	CSRR[0:1]	Watchdog timeout: as specified in *Book E: Enhanced PowerPC™ Architecture, Rev 1.0*, Ch. 8, pg. 196–197 and in the *e200z6 PowerPC™ Core Reference Manual,* Rev 0.
Data TLB error	IVOR 13	—	SRR[0:1]	Data translation lookup did not match a valid entry in the TLB
Instruction TLB error	IVOR 14	—	SRR[0:1]	Instruction translation lookup did not match a valid TLB entry

Table 13-12. Interrupts and Conditions (continued)

Interrupt Type	Interrupt Vector Offset Register	Enables[1]	Core Register in Which State Information is Saved	Causing Conditions
Debug	IVOR 15	DE, IDM	CSSR[0:1]	Debugger when HIDO[DAPUEN] = 0. Caused by trap, instruction address compare, data address compare, instruction complete, branch taken, return from interrupt, interrupt taken, debug counter, external debug event, unconditional debug event
		DE, IDM	DSRR[0:1]	Debugger when HIDO[DAPUEN] = 1, and caused by same conditions as above.
Reserved	IVOR 16–31			
SPE unavailable exception	IVOR 32	—	SRR[0:1]	SPE APU instruction when MSR[SPE] = 0, and see Section 5.6.18, SPE APU Unavailable Interrupt, in the *e200z6 PowerPC™ Core Reference Manual,* Rev 0.
SPE data exception	IVOR 33	—	SRR[0:1]	SPE FP data exception and see Section 5.6.19, SPE Floating-Point Data Interrupt, in the *e200z6 PowerPC™ Core Reference Manual,* Rev 0.
SPE round exception	IVOR 34	—	SRR[0:1]	Inexact result from floating-point instruction. See Section 5.6.2, SPE Floating-Point Round Interrupt, in the *e200z6 PowerPC™ Core Reference Manual,* Rev 0.

1 CE, ME, EE, DE are in the MSR. DIE, FIE, and WIE are in the TCR. “src” signifies the individual enable for each INTC source. The debug interrupt, IVOR 15, also requires EDM = 0 (EDM and IDM are in the DBCR0 register).

2 Autovectored External and Critical Input interrupts use this IVOR. Vectored interrupts supply an interrupt vector offset directly.

13.3.4 Bus Interface Unit (BIU)

The BIU encompasses control and data signals supporting instruction and data transfers. A data bus width of 64 bits is implemented. The memory interface supports read and write transfers of 8, 16, 24, 32, and 64 bits, supports burst transfers of four doublewords, and operates in a pipelined fashion.

Single-beat transfers are supported for cache-inhibited read and write cycles, and write-buffer writes. Burst transfers of four doublewords are supported for cache linefill and copyback operations.

13.3.5 Timer Facilities

The core provides a set of registers to provide fixed interval timing and watchdog functions for the system. All of these must be initialized during start-up. The registers associated with fixed interval timer and watchdog functions are the following:

- Timer control register (TCR)—provides control of the timer and watchdog facilities.
- Timer status register (TSR)—provides status of the timer facilities.
- Time base registers (TBU and TBL)—two 32-bit registers (upper and lower) that are concatenated to provide a long-period, 64-bit counter.

- Decrementer register (DEC)—a decrementing counter that is updated at the same rate as the time base. The DEC provides a means of signaling an exception after a specified amount of time. The DEC is typically used as a general-purpose software timer. Note that the decrementer always runs when the system is clocked, and can be written to by software at any time.
- Decrementer auto reload register (DECAR)—provides a value that is automatically reloaded (if enabled) into the decrementer register when the decrementer reaches 0.

For more information on the fixed-interval timer, watchdog timer, and timer and counter registers, refer to the *e200z6 PowerPC™ Core Reference Manual* and *EREF: A Programmer's Reference Manual for Freescale Book E Processors.*

13.3.6 Signal Processing Extension APU (SPE APU)

13.3.6.1 Overview

The Power Architecture embedded category 32-bit instructions operate on the lower (least significant) 32 bits of the 64-bit GPRs. New SPE instructions are defined that view the 64-bit register as being composed of a vector of two 32-bit elements, and some of the instructions also read or write 16-bit elements. These new instructions can also be used to perform scalar operations by ignoring the results of the upper 32-bit half of the register file.

Some instructions are defined that produce a 64-bit scalar result. Vector fixed-point instructions operate on a vector of two 32-bit or four 16-bit fixed-point numbers resident in the 64-bit GPRs. Vector floating-point instructions operate on a vector of two 32-bit single-precision floating-point numbers resident in the 64-bit GPRs. Scalar floating-point instructions operate on the lower half of GPRs. These single-precision floating-point instructions do not have a separate register file; there is a single shared register file for all instructions. Figure 13-18 shows two different representations of the 64-bit GPRs. The shaded half is the only region operated on by the 32-bit Power Architecture embedded category instructions.

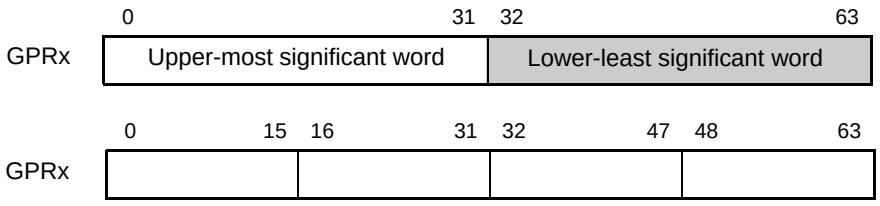


Figure 13-18. 64-bit General-Purpose Registers

13.3.7 SPE Programming Model

Not all SPE instructions record events such as overflow, saturation, and negative/positive result. See the description of the individual SPE instruction in the e200z6 core reference for information on which conditions are recorded and where they are recorded. Most SPE instructions record conditions to the SPEFSCR. Vector compare instructions store the result of the comparison into the condition register (CR).

The e200z6 core has a 64-bit architectural accumulator register that holds the results of the SPE multiply accumulate (MAC) fixed-point instructions. The accumulator allows back-to-back execution of dependent fixed-point MAC instructions, something that is found in the inner loops of DSP code such as filters. The accumulator is partially visible to the programmer in that its results do not have to be explicitly read to use them. Instead, they are always copied into a 64-bit destination GPR specified as part of the instruction. The accumulator however, has to be explicitly cleared when starting a new MAC loop. Based on the type of instruction, the accumulator can hold either a single 64-bit value or a vector of two 32-bit elements.

13.3.8 12.3.8 Wait Instruction

The z650n3e implements support for the wait instruction. Executing the wait instruction stops synchronous processor activity. Executing a wait instruction ensures that all instructions have completed before the wait instruction completes, causes processor instruction fetching to cease, and ensures that no subsequent instructions are initiated until an interrupt or a debug interrupt occurs. Once the wait instruction has completed, the program counter will point to the next sequential instruction. The saved value in xSRR0 when the processor re-initiates activity will point to the instruction following the wait instruction.

Execution of a wait instruction places the e200z6 in the “waiting” state. It can be used for power reduction in a interrupt based system when the core has no processing tasks. An internal output signal from the core indicates this to the CRP module that the core has entered the waiting state. This is used by the CRP to place the SOC into low power mode if it has been requested by the user.

When in the “waiting” state, the clock to the core continues to run only if other crossbar masters are active.

13.4 Power Architecture Instruction Extensions – VLE

The variable length encoding (VLE) provides an extension to 32-bit Power Architecture. There are additional operations defined using an alternate instruction encoding to enable reduced code footprint. This alternate encoding set is selected on an instruction page basis. A single page attribute bit selects between standard Power Architecture instruction encodings and VLE instructions for that page of memory. This page attribute is an extension to the Power Architecture page attributes. Pages can be freely intermixed, allowing for a mixture of code using both types of encodings.

Instruction encodings in pages marked as using the VLE extension are either 16 or 32 bits long, and are aligned on 16-bit boundaries. Therefore, all instruction pages marked as VLE are required to use big-endian byte ordering.

This section describes the various extensions to the Power Architecture instructions that support the VLE extension.

rfci, **rfdi**, **rfi**Not the mask bit 62 of CSRR0, DSRR0, or SRR0 respectively.
The destination address is [D,C]SRR0[32:62] || 0b0.

bclr, **bclrl**, **bcctr**, **bcctrl**Not the mask bit 62 of the LR or CTR respectively.
The destination address is [LR,CTR][32:62] || 0b0.

13.5 External References

In addition to the Power Architecture instructions, the device supports e200z6 core-specific instructions and SPE APU instructions and VLE instructions. For further information see the following documents:

- *e200z6 PowerPC™ Core Reference Manual*
- *PowerPC™ Microprocessor Family: The Programming Environment for 32-bit Microprocessors*
- *Book E: Enhanced PowerPC™ Architecture*
- *EREF: A Programmer's Reference Manual for Freescale Book E Processors*
- *VLEPIM: Variable Length Encoding (VLE) Extension Programming Interface Manual*
- Addendum to e200z6 PowerPC™ Core Reference Manual: e200z6 with VLE
- Errata to e200z6 PowerPC™ Core Reference Manual, Rev. 0

Chapter 14
e200z0 Core (Z0)

14.1 Introduction

The e200 processor family is a set of CPU cores that implement low-cost versions of the Power Architecture Book E architecture. e200 processors are designed for deeply embedded control applications that require low cost solutions rather than maximum performance.

The e200z0 processors integrate an integer execution unit, branch control unit, instruction fetch and load/store units, and a multi-ported register file capable of sustaining three read and two write operations per clock. Most integer instructions execute in a single clock cycle.

The e200z0 core is a single-issue, 32-bit Power Architecture Book E VLE-only design with 32-bit general purpose registers (GPRs). All arithmetic instructions that execute in the core operate on data in the general purpose registers (GPRs).

NOTE

On the PXN20 family, the e200z0 core runs at half the system clock frequency. Unless otherwise noted in this chapter, all stated clock delays are relative to the e200z0 core clock, not the system clock.

Instead of the base Power Architecture Book E instruction set support, the e200z0 core implements only the VLE (variable-length encoding) APU, providing improved code density. The VLE APU is further described in *PowerPC VLE APU Definition, Version 1.01*, a separate document.

In the remainder of this document, the e200z0 core is also referred to as the “e200z0” core or “e200 core.”

14.1.1 Features

The following is a list of some of the key features of the e200z0 core:

- 32-bit Power Architecture Book E VLE-only programmer’s model
- Single issue, 32-bit CPU
- Implements the VLE APU for reduced code footprint
- In-order execution and retirement
- Precise exception handling
- Branch processing unit
  - — Dedicated branch address calculation adder
- Supports instruction and data access via a unified 32-bit Instruction/Data BIU (e200z0 only).
- Load/store unit

  - 1 cycle load latency
  - Fully pipelined
  - Big-endian support only
  - Misaligned access support
  - Zero load-to-use pipeline bubbles for aligned transfers
- Power management
  - Low power design
  - Power saving modes: doze, nap, sleep, and wait
  - Dynamic power management of execution units

NOTE

The PXN20 does not use the core's HID0[DOZE,NAP,SLEEP] bits to enter/exit low-power modes. Entry to and exit from low-power modes is managed by the CRP module.

14.2 Microarchitecture Summary

The execution pipeline four stages operate in an overlapped fashion, allowing single-clock instruction execution for most instructions. These stages are as follows:

1. The instruction fetch
2. Instruction decode/register file read/effective address calculation
3. Execute/memory access
4. Register writeback

The integer execution unit consists of a 32-bit Arithmetic Unit (AU), a Logic Unit (LU), a 32-bit Barrel shifter (Shifter), a Mask-Insertion Unit (MIU), a Condition Register manipulation Unit (CRU), a Count-Leading-Zeros unit (CLZ), an 8x32 Hardware Multiplier array, result feed-forward hardware, and a hardware divider.

Arithmetic and logical operations are executed in a single cycle with the exception of the divide and multiply instructions. A count-leading-zeros unit operates in a single clock cycle.

The Instruction Unit contains a PC incrementer and a dedicated Branch Address adder to minimize delays during change of flow operations. Sequential prefetching is performed to ensure a supply of instructions into the execution pipeline. Prefetched instructions are placed into an instruction buffer with 2 entries, each capable of holding a single 32-bit instruction or a pair of 16-bit instructions.

Conditional branches which are not taken execute in a single clock. All taken branches have an execution time of two clocks.

Memory load and store operations are provided for byte, halfword, and word (32-bit) data with automatic zero or sign extension of byte and halfword load data as well as optional byte reversal of data. These instructions can be pipelined to allow effective single cycle throughput. Load and store multiple word instructions allow low overhead context save and restore operations. The load/store unit contains a

dedicated effective address adder to allow effective address generation to be optimized. Also, a load-to-use dependency does not incur any pipeline bubbles for most cases.

The Condition Register unit supports the condition register (CR) and condition register operations defined by the Power Architecture. The condition register consists of eight 4-bit fields that reflect the results of certain operations, such as move, integer and floating-point compare, arithmetic, and logical instructions, and provide a mechanism for testing and branching.

Vectored and autovectored interrupts are supported by the CPU. Vectored interrupt support is provided to allow multiple interrupt sources to have unique interrupt handlers invoked with no software overhead.

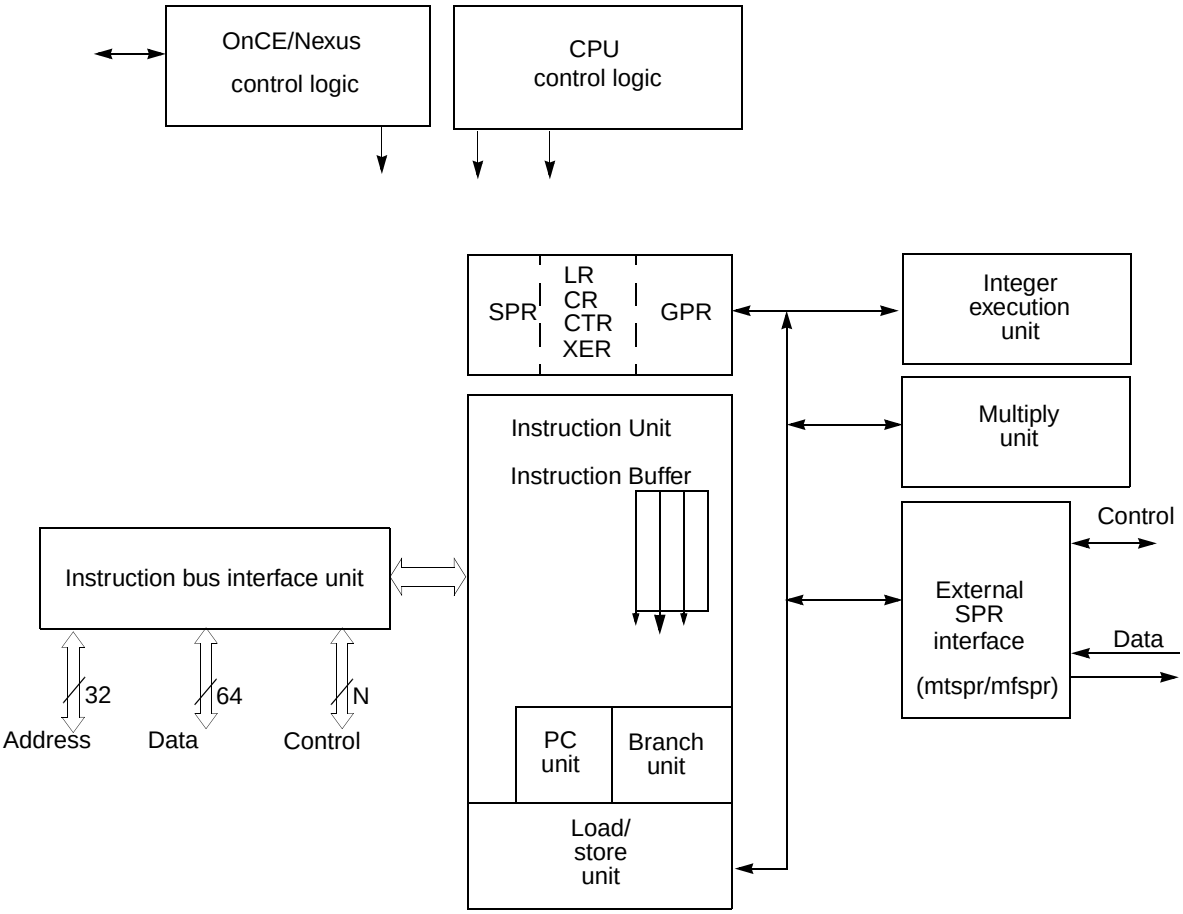


Figure 14-1. e200z0 Block Diagram

14.2.1 Instruction Unit Features

The features of the e200 Instruction unit are:

- 32-bit instruction fetch path supports fetching of one 32-bit instruction per clock, or as many as two 16-bit VLE instructions per clock.
- Instruction buffer with two entries, each holding a single 32-bit instruction, or a pair of 16-bit instructions

- Dedicated PC incrementer supporting instruction prefetches
- Branch unit with dedicated branch address adder supporting single cycle of execution of certain branches, two cycles for all others

14.2.2 Integer Unit Features

The e200 integer unit supports single cycle execution of most integer instructions:

- 32-bit AU for arithmetic and comparison operations
- 32-bit LU for logical operations
- 32-bit priority encoder for count leading zero's function
- 32-bit single cycle barrel shifter for shifts and rotates
- 32-bit mask unit for data masking and insertion
- Divider logic for signed and unsigned divide in 5 to 34 clocks with minimized execution timing
- 8x32 hardware multiplier array supports 1 to 4 cycle 32x32→32 multiply (early out)

14.2.3 Load/Store Unit Features

The e200 load/store unit supports load, store, and the load multiple / store multiple instructions:

- 32-bit effective address adder for data memory address calculations
- Pipelined operation supports throughput of one load or store operation per cycle
- 32-bit interface to memory

14.2.4 e200z0 System Bus Features

The features of the e200z0 System Bus interface are as follows:

- Unified Instruction/Data Bus
- 32-bit address bus plus attributes and control
- Separate uni-directional 32-bit read data bus and 32-bit write data bus
- Overlapped, in-order accesses

14.2.5 Nexus 2+ Features

The Nexus 2+ module is compliant with Class 2 of the IEEE®-ISTO 5001-2003 standard, with additional Class 3 and Class 4 features available. The following features are implemented:

- Program trace via branch trace messaging (BTM). Branch trace messaging displays program flow discontinuities (direct and indirect branches, exceptions, etc.), allowing the development tool to interpolate what transpires between the discontinuities. Thus, static code may be traced.
- Ownership trace via ownership trace messaging. (OTM facilitates ownership trace by providing visibility of which process ID or operating system task is activated. An ownership trace message is transmitted when a new process/task is activated, allowing the development tool to trace ownership flow.)

- Run-time access to the processor memory map via the JTAG port. This allows for enhanced download/upload capabilities.
- Watchpoint messaging through the auxiliary interface.
- Watchpoint trigger enable of program trace messaging.
- Auxiliary interface for higher data input/output (Nexus interface shared with Z6 core).
  - 12 message data out pins (MDO[11:0])
  - Two message start/end out pins ($\overline{\text{MSEO}}$[1:0])
  - One watchpoint event pin ($\overline{\text{EVTO}}$)
  - One event in pin ($\overline{\text{EVTI}}$)
  - One message clock out (MCKO) pin
- Registers for program trace, ownership trace, and watchpoint trigger control.
- All features controllable and configurable via the JTAG port.

14.3 Core Registers and Programmer's Model

This section describes the registers implemented in the e200z0 core. It includes an overview of registers defined by the Power Architecture Book E architecture, highlighting differences in how these registers are implemented in the e200 core, and provides a detailed description of e200-specific registers. Full descriptions of the architecture-defined register set are provided in Power Architecture Book E Specification.

The Power Architecture Book E defines register-to-register operations for all computational instructions. Source data for these instructions are accessed from the on-chip registers or are provided as immediate values embedded in the opcode. The three-register instruction format allows specification of a target register distinct from the two source registers, thus preserving the original data for use by other instructions. Data is transferred between memory and registers with explicit load and store instructions only.

Figure 14-2 and Figure 14-3 show the e200 register set including the registers which are accessible while in supervisor mode, and the registers which are accessible in user mode. The number to the right of the special-purpose registers (SPRs) is the decimal number used in the instruction syntax to access the register (for example, the integer exception register (XER) is SPR 1).

NOTE

e200z0 is a 32-bit implementation of the Power Architecture Book E specification. In this document, register bits are sometimes numbered from bit 0 (Most Significant Bit) to 31 (Least Significant Bit), rather than the Book E numbering scheme of 32:63, thus register bit numbers for some registers in Book E are 32 higher.

Where appropriate, the Book E defined bit numbers are shown in parentheses.

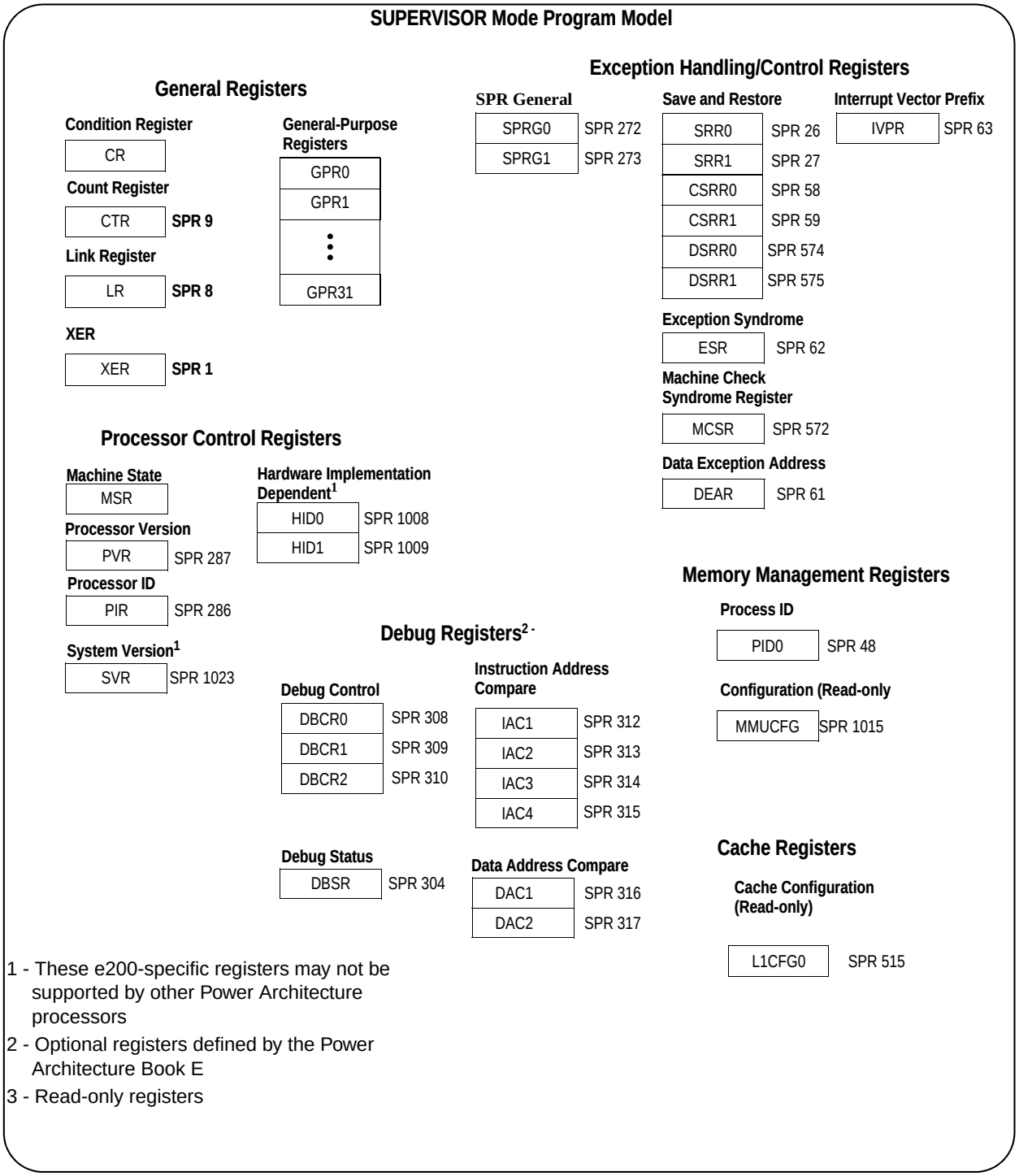


Figure 14-2. e200z0 Supervisor Mode Programmer's Model

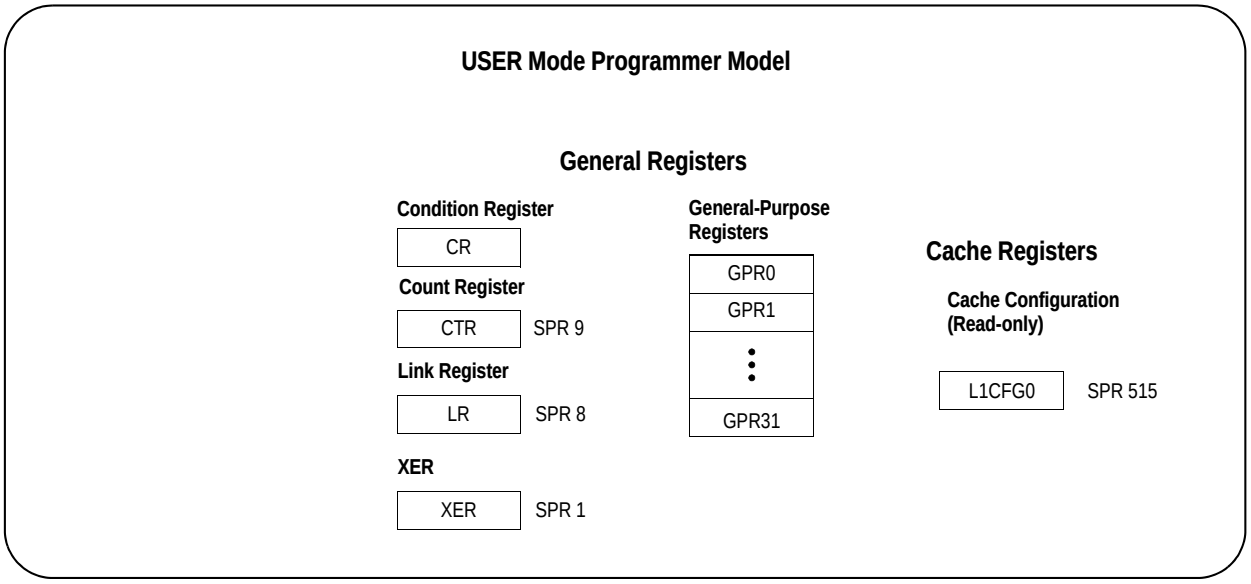


Figure 14-3. e200 User Mode Program Model

General purpose registers (GPRs) are accessed through instruction operands. Access to other registers can be explicit (by using instructions for that purpose such as Move to Special Purpose Register (**mtspr**) and Move from Special Purpose Register (**mfspr**) instructions) or implicit as part of the execution of an instruction. Some registers are accessed both explicitly and implicitly.

14.3.1 Power Architecture Book E Registers

e200 supports a subset of the registers defined by *Power Architecture™ Book E Specification*. Notable exceptions are the Floating Point registers FPR0-FPR31 and FPSCR. e200z0 does not support the Book E floating-point architecture. The e200-supported Power Architecture Book E registers are described as follows (e200-specific registers are described in the Section 14.3.2, e200-Specific Special Purpose Registers):

14.3.1.1 User-Level Registers

The user-level registers can be accessed by all software with either user or supervisor privileges. They include the following:

- General-purpose registers (GPRs). The thirty-two 32-bit GPRs (GPR0–GPR31) serve as data source or destination registers for integer instructions and provide data for generating addresses.
- Condition register (CR). The 32-bit CR consists of eight 4-bit fields, CR0–CR7, that reflect results of certain arithmetic operations and provide a mechanism for testing and branching. See Condition Register (CR), in Chapter 3, Branch and Condition Register Operations, of *Power Architecture Book E Specification*.

  The remaining user-level registers are SPRs. Note that the Power Architecture Book E provides the **mtspr** and **mfspr** instructions for accessing SPRs.

Integer exception register (XER). The XER indicates overflow and carries for integer operations. See XER Register (XER), in Chapter 4, nteger Operations, of *Power Architecture Book E Specification* for more information.

- Link register (LR). The LR provides the branch target address for the Branch to Link Register (**se_blr**, **se_blrl**) instructions, and is used to hold the address of the instruction that follows a branch and link instruction, typically used for linking to subroutines. See Link Register (LR), in Chapter 3, Branch and Condition Register Operations, of *Power Architecture Book E Specification*.
- Count register (CTR). The CTR holds a loop count that can be decremented during execution of appropriately coded branch instructions. The CTR also provides the branch target address for the Branch to Count Register (**se_bctr**, **se_bctrl**) instructions. See Count Register (CTR), in Chapter 3, Branch and Condition Register Operations, of *Power Architecture Book E Specification*.

14.3.1.2 Supervisor-Level Registers

In addition to the registers accessible in user mode, Supervisor-level software has access to additional control and status registers used for configuration, exception handling, and other operating system functions. The Power Architecture Book E defines the following supervisor-level registers:

- Processor Control Registers
  - Machine State Register (MSR). The MSR defines the state of the processor. The MSR can be modified by the Move to Machine State Register (**mtmsr**), System Call (**se_sc**), and Return from Exception (**se_rfi**, **se_rfci**, **se_rfdi)** instructions. It can be read by the Move from Machine State Register (**mfmsr)** instruction. When an interrupt occurs, the contents of the MSR are saved to one of the machine state save/restore registers (SRR1, CSRR1, DSRR1).
  - Processor version register (PVR). This register is a read-only register that identifies the processor type and version (model) and the revision level of the processor. Table 14-1 shows the PVR values and the corresponding processor type and version numbers for the cores used on the PXN20 family.

Table 14-1. PVR Values, and Processor Type and Version Numbers

Device	Core	PVR Value	Type	Version
PXN20	e200z0	0x8171_0000	0x17	0x1

  - Processor Identification Register (PIR). This read-only register is provided to distinguish the processor from other processors in the system.
- Storage Control Register
  - Process ID Register (PID, also referred to as PID0). This register is provided to indicate the current process or task identifier. It is used by the Nexus2 module for Ownership Trace message generation. Although the Power Architecture Book E allows for multiple PIDs, e200z0 implements only one.
- Interrupt Registers
  - Data Exception Address Register (DEAR). After most Data Storage Interrupts (DSI), or on an Alignment Interrupt, the DEAR is set to the effective address (EA) generated by the faulting instruction.

- SPRG0–SPRG1. The SPRG0–SPRG1 registers are provided for operating system or interrupt handler use.
- Exception Syndrome Register (ESR). The ESR register provides a syndrome to differentiate between the different kinds of exceptions which can generate the same interrupt.
- Interrupt Vector Prefix Register (IVPR). This register together with hardwired offsets which replace the IVOR0-15 registers provide the address of the interrupt handler for different classes of interrupts.
- Save/Restore Register 0 (SRR0). The SRR0 register is used to save machine state on a non-critical interrupt, and contains the address of the instruction at which execution resumes when an **se_rfi** instruction is executed at the end of a non-critical class interrupt handler routine.
- Critical Save/Restore Register 0 (CSRR0). The CSRR0 register is used to save machine state on a critical interrupt, and contains the address of the instruction at which execution resumes when an **se_rfci** instruction is executed at the end of a critical class interrupt handler routine.
- Save/Restore Register 1 (SRR1). The SRR1 register is used to save machine state from the MSR on non-critical interrupts, and to restore machine state when **se_rfi** executes.
- Critical Save/Restore Register 1 (CSRR1). The CSRR1 register is used to save machine state from the MSR on critical interrupts, and to restore machine state when **se_rfci** executes.

- Debug Facility Registers
  - Debug Control Registers (DBCR0–DBCR2). These registers provide control for enabling and configuring debug events.
  - Debug Status Register (DBSR). This register contains debug event status.
  - Instruction Address Compare registers (IAC1–IAC4). These registers contain addresses and/or masks which are used to specify Instruction Address Compare debug events.
  - Data Address Compare registers (DAC1–2). These registers contain addresses and/or masks which are used to specify Data Address Compare debug events.
  - e200 does **not** implement the Data Value Compare registers (DVC1 and DVC2).

14.3.2 e200-Specific Special Purpose Registers

The Power Architecture Book E architecture allows implementation-specific special purpose registers. Those incorporated in the e200 core are as follows:

14.3.2.1 User-Level Registers

The user-level registers can be accessed by all software with either user or supervisor privileges. They include the following:

- The L1 Cache Configuration register (L1CFG0). This read-only register allows software to query the configuration of the L1 Cache. For the e200z0, this register returns all zeros indicating no cache is present.

14.3.2.2 Supervisor-Level Registers

The following supervisor-level registers are defined in e200 in addition to the Power Architecture Book E registers described above:

- Configuration Registers
  - Hardware implementation-dependent register 0 (HID0). This register controls various processor and system functions.
  - Hardware implementation-dependent register 1 (HID1). This register controls various processor and system functions.
- Exception Handling and Control Registers
  - Machine Check Syndrome register (MCSR). This register provides a syndrome to differentiate between the different kinds of conditions which can generate a Machine Check.
  - Debug Save/Restore register 0 (DSRR0). When enabled, the DSRR0 register is used to save the address of the instruction at which execution continues when **se_rfdi** executes at the end of a debug interrupt handler routine.
  - Debug Save/Restore register 1 (DSRR1). When enabled, the DSRR1 register is used to save machine status on debug interrupts and to restore machine status when **se_rfdi** executes.
- L1 Cache Configuration Register (L1CFG0) is a read-only register that allows software to query the configuration of the L1 Cache. For the e200z0, this register returns all zeros.
- System version register (SVR). This register is a read-only register that identifies the version (model) and revision level of the device which includes an e200 Power Architecture processor.

Note that it is not guaranteed that the implementation of e200 core-specific registers is consistent among Power Architecture processors, although other processors may implement similar or identical registers. All e200 SPR definitions are compliant with the Freescale EIS specification definitions.

14.3.3 e200z0 Core Complex Features not Supported on the PXN20

The PXN20 implements a subset of the e200z0 core complex features. The e200z0 core complex features that are not supported in the PXN20 are described in Table 14-2.

Table 14-2. e200z0 Features Not Supported on the PXN20

Description	Function/Category
The less significant halfword of the Processor Version Register (PVR) provides the revision level which is comprised of the following three bit fields: Reserved = 0x00 Revision = 0x0 ID = 0x0 The more significant halfword of the Processor Version Register (PVR) provides the processor type and version number (see Table 14-1).	PVR Value
Nexus registers are not accessible by code running in User or Supervisor mode. Nexus registers can be accessed only by external tools via the Nexus port.	Debug

14.4 Interrupt Types

The interrupts implemented on the PXN20 and the exception conditions that cause them are listed in Table 14-3.

Table 14-3. Exceptions and Conditions

Interrupt Type	Interrupt Vector Offset Register	Causing Conditions
System reset	none, vector to address determined by CRP_Z0VEC	1. Reset. 2. Debug Reset Control.
Critical Input	IVOR 0[1]	Non-maskable interrupt request and MSR[CE] = 1.
Machine check	IVOR 1	1. Machine check error and MSR[ME] = 1. 2. Bus error (XTE) with MSR[EE] = 0 and current MSR[ME] = 1
Data Storage	IVOR 2	1. Access control. (unused on e200z0) 2. Precise external termination error and MSR[EE] = 1.
Instruction Storage	IVOR 3	1. Access control. (unused on e200z0) 2. Precise external termination error and MSR[EE] = 1.
External Input	IVOR 4[1]	Interrupt request and MSR[EE] = 1.
Alignment	IVOR 5	1. **lmw**, **stmw** not word aligned. 2. **lwarx** or **stwcx.** not word aligned.
Program	IVOR 6	Illegal, Privileged, Trap, Unimplemented Operation.
Floating-point unavailable	IVOR 7	Unused
System call	IVOR 8	Execution of the System Call (**se_sc**) instruction

Table 14-3. Exceptions and Conditions (continued)

Interrupt Type	Interrupt Vector Offset Register	Causing Conditions
AP unavailable	IVOR 9	Unused
Decrementer	IVOR 10	Unused
Fixed Interval Timer	IVOR 11	Unused
Watchdog Timer	IVOR 12	Unused
Data TLB Error	IVOR 13	Unused
Instruction TLB Error	IVOR 14	Unused
Debug	IVOR 15	Trap, Instruction Address Compare, Data Address Compare, Instruction Complete, Branch Taken, Return from Interrupt, Interrupt Taken, External Debug Event, Unconditional Debug Event
Reserved	IVOR 16-31	—

[1] Autovectored External and Critical Input interrupts use this IVOR. Vectored interrupts supply an interrupt vector offset directly.

Table 14-4 summarizes the e200z0 interrupts. Each ISR begins at a fixed offset as defined below.

Table 14-4. e200z0 Interrupts

IRQ #	Offset	Size [Byte]	Resource
—	0x0000	16	Critical Input (NMI)
—	0x0010	16	Machine check
—	0x0020	16	Data Storage
—	0x0030	16	Instruction Storage
—	0x0040	16	External Input (INTC software vector mode)
—	0x0050	16	Alignment
—	0x0060	16	Program
—	0x0070	16	Reserved
—	0x0080	16	System call
—	0x0090	96	Unused
—	0x00F0	16	Debug
—	0x0100	1792	Unused

14.5 Bus Interface Unit (BIU)

The BIU encompasses control and data signals supporting instruction and data transfers, support for interrupts, including vectored interrupt logic, reset support, power management interface signals, debug event signals, processor state information, Nexus /OnCE / JTAG interface signals, and a test interface.

The memory portion of the e200 core interface is comprised of a 32-bit wide system bus and a unified bus. The memory interface supports read and write transfers of 8, 16, 24, and 32 bits, supports misaligned transfers, and operates in a pipelined fashion.

Single-beat and misaligned transfers are supported for read and write cycles. Incrementing burst transfers are supported for instruction prefetch operations.

Chapter 15
Semaphores

15.1 Introduction

In a dual processor chip, semaphores are used to let each processor know who has control of common memory. Before a core can update or read memory coherently, it has to check the semaphore to see if the other core is not already updating the memory. If the semaphore is clear, it can write common memory, but if it is set, it has to wait for the other core to finish and clear the semaphore.

The semaphores module provides the hardware support needed in multi-core systems for implementing semaphores and provide a simple mechanism to achieve lock/unlock operations via a single write access. This approach eliminates architecture-specific implementations like atomic (indivisible) read-modify-write instructions or reservation mechanisms. The result is an architecture-neutral solution that provides hardware-enforced gates as well as other useful system functions related to the gating mechanisms.

15.1.1 Block Diagram

Figure 15-1 is a simplified block diagram of the semaphores module that illustrates the functionality and interdependence of major blocks. In the diagram, the register blocks named gate0, gate1, ..., gate 15 include the finite state machines implementing the semaphore gates plus the interrupt notification logic.

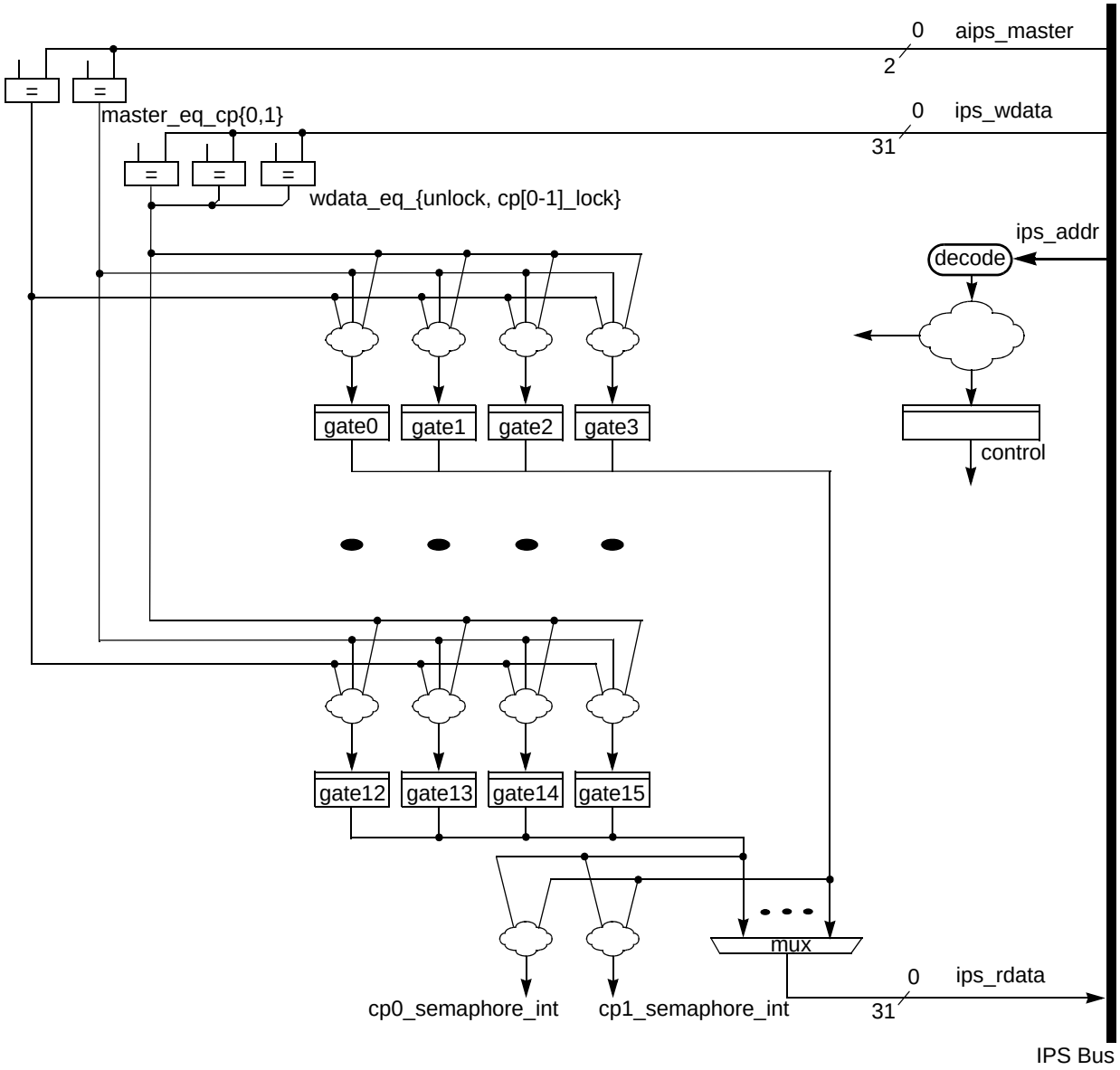


Figure 15-1. Semaphores Block Diagram

15.1.2 Features

The semaphores module implements hardware-enforced semaphores as a peripheral device and has these major features:

- Support for 16 hardware-enforced gates in a dual-processor configuration
  - Each hardware gate appears as a three-state, 2-bit state machine, with all 16 gates mapped as an array of bytes
    - Three-state implementation

      if gate = 0b00, then state = unlocked

if gate = 0b01, then state = locked by e200z6 (master ID = 0)

if gate = 0b10, then state = locked by e200z0 (master ID = 1)

- Uses the bus master ID number as a reference attribute plus the specified data patterns to validate all write operations
- After it is locked, the gate must be unlocked by a write of zeroes from the locking processor

— Optionally enabled interrupt notification after a failed lock write provides a mechanism to indicate the gate is unlocked

— Secure reset mechanisms are supported to clear the contents of individual semaphore gates or notification logic, and clear_all capability

NOTE

Semaphore gates that are locked when entering sleep mode are cleared by the internal reset generated when exiting sleep mode.

15.1.3 Modes of Operation

The semaphores module does not support any special modes of operation.

15.2 Signal Description

The semaphores module does not include any external signals.

15.3 Memory Map and Registers

This section provides a detailed description of all semaphores registers.

15.3.1 Module Memory Map

The semaphores programming model map is shown in Table 15-1. The address of each register is given as an offset to the semaphore base address. Registers are listed in address order, identified by complete name and mnemonic, and list the type of accesses allowed.

Table 15-1. Semaphores Memory Map

Offset from SEMA4_BASE (0xFFF1_0000)	Register	Access	Reset Value	Section/ Page
0x0000	SEMA4_Gate00—Semaphores gate 0	R/W	0x00	15.3.2.1/15-4
0x0001	SEMA4_Gate01—Semaphores gate 1	R/W	0x00	15.3.2.1/15-4
0x0002	SEMA4_Gate02—Semaphores gate 2	R/W	0x00	15.3.2.1/15-4
0x0003	SEMA4_Gate03—Semaphores gate 3	R/W	0x00	15.3.2.1/15-4
0x0004	SEMA4_Gate04—Semaphores gate 4	R/W	0x00	15.3.2.1/15-4
0x0005	SEMA4_Gate05—Semaphores gate 5	R/W	0x00	15.3.2.1/15-4
0x0006	SEMA4_Gate06—Semaphores gate 6	R/W	0x00	15.3.2.1/15-4

Table 15-1. Semaphores Memory Map (continued)

Offset from SEMA4_BASE (0xFFF1_0000)	Register	Access	Reset Value	Section/ Page
0x0007	SEMA4_Gate07—Semaphores gate 7	R/W	0x00	15.3.2.1/15-4
0x0008	SEMA4_Gate08—Semaphores gate 8	R/W	0x00	15.3.2.1/15-4
0x0009	SEMA4_Gate09—Semaphores gate 9	R/W	0x00	15.3.2.1/15-4
0x000A	SEMA4_Gate10—Semaphores gate 10	R/W	0x00	15.3.2.1/15-4
0x000B	SEMA4_Gate11—Semaphores gate 11	R/W	0x00	15.3.2.1/15-4
0x000C	SEMA4_Gate12—Semaphores gate 12	R/W	0x00	15.3.2.1/15-4
0x000D	SEMA4_Gate13—Semaphores gate 13	R/W	0x00	15.3.2.1/15-4
0x000E	SEMA4_Gate14—Semaphores gate 14	R/W	0x00	15.3.2.1/15-4
0x000F	SEMA4_Gate15—Semaphores gate 15	R/W	0x00	15.3.2.1/15-4
0x0010–0x003F	Reserved			
00x040	SEMA4_CP0INE—Semaphores CP0 IRQ notification enable	R/W	0x0000	15.3.2.2/15-5
0x0042–0x0047	Reserved			
0x0048	SEMA4_CP1INE—Semaphores CP1 IRQ notification enable	R/W	0x0000	15.3.2.2/15-5
0x004A–0x07F	Reserved			
0x0080	SEMA4_CP0NTF—Semaphores CP0 IRQ notification	R	0x0000	15.3.2.3/15-6
0x008 2–00x087	Reserved			
0x0088	SEMA4_CP1NTF—Semaphores CP1 IRQ notification	R	0x0000	15.3.2.2/15-5
0x008A–0x00FF	Reserved			
0x0100	SEMA4_RSTGT—Semaphores reset gate	R/W	0x0000	15.3.2.4/15-6
0x0102	Reserved			
0x0104	SEMA4_RSTNTF—Semaphores reset IRQ notification	R/W	0x00000	15.3.2.5/15-8
0x0106–0x3FFF	Reserved			

15.3.2 Register Descriptions

This section lists the semaphores registers in address order and describes the registers and their bit fields.

15.3.2.1 Semaphores Gate *n* Register (SEMA4_GATE*n*)

Each semaphore gate is implemented in a 2-bit finite state machine, right-justified in a byte data structure. The hardware uses the bus master number in conjunction with the data patterns to validate all attempted write operations. Only processor bus masters can modify the gate registers. After it is locked, a gate must be opened (unlocked) by the locking processor core.

Multiple gate values can be read in a single access, but only a single gate at a time can be updated via a write operation. 16- and 32-bit writes to multiple gates are allowed, but the write data operand must update the state of a single gate only. A byte write data value of 0x03 is defined as no operation and does not affect the state of the corresponding gate register. Attempts to write multiple gates in a single-aligned access with a size larger than an 8-bit (byte) reference generate an error termination and do not allow any gate state changes.

Offset: SEMA4_BASE + *n* (*n* = 0, 1, 2,..., 15) Access: User read/write

	0	1	2	3	4	5	6	7
R	0	0	0	0	0	0	GTFSM	
W								
Reset	0	0	0	0	0	0	0	0

Figure 15-2. SEMA4 Gate *n* Register (SEMA4_GATE*n*)

Table 15-2. SEMA4_GATE*n* Field Descriptions

Field	Description
GTFSM	Gate Finite State Machine. The hardware gate is maintained in a three-state implementation, defined as: 00 The gate is unlocked (free). 01 The gate has been locked by processor 0. 10 The gate has been locked by processor 1. 11 This state encoding is never used and therefore reserved. Attempted writes of 0x03 are treated as no operation and do not affect the gate state machine. **Note:** The state of the gate reflects the last processor that locked it, which can be useful during system debug.

15.3.2.2 Semaphores Processor *n* IRQ Notification Enable (SEMA4_CP{0,1}INE)

The application of a hardware semaphore module provides an opportunity for implementation of helpful system-level features. An example is an optional mechanism to generate a processor interrupt after a failed lock attempt. Traditional software gate functions execute a spin-wait loop in an effort to obtain and lock the referenced gate. With this module, the processor that fails in the lock attempt could continue with other tasks and allow a properly-enabled notification interrupt to return its execution to the original lock function.

The optional notification interrupt function consists of two registers for each processor: an interrupt notification enable register (SEMA4_CP*n*INE) and the interrupt request register (SEMA4_CP*n*NTF). To support implementations with more than 16 gates, these registers can be referenced with aligned 16- or 32-bit accesses. For the SEMA4_CP*n*INE registers, unimplemented bits read as zeroes and writes are ignored.

Offset: SEMA4_BASE + 0x0040 (SEMA4_CP0INE) Access: User read/write
SEMA4_BASE + 0x0048 (SEMA4_CP1INE)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	INE0	INE1	INE2	INE3	INE4	INE5	INE6	INE7	INE8	INE9	INE10	INE11	INE12	INE13	INE14	INE15
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 15-3. Semaphores Processor *n* IRQ Notification Enable (SEMA4_CP{0,1}INE)

Table 15-3. SEMA4_CP{0,1}NTF Field Descriptions

Field	Description
INE*n*	Interrupt Request Notification Enable *n*. This field is a bitmap to enable the generation of an interrupt notification from a failed attempt to lock gate *n*. 0 The generation of the notification interrupt is disabled. 1 The generation of the notification interrupt is enabled.

15.3.2.3 Semaphores Processor *n* IRQ Notification (SEMA4_CP{0,1}NTF)

The notification interrupt is generated via a unique finite state machine, one per hardware gate. This machine operates in the following manner:

- When an attempted lock fails, the FSM enters a first state where it waits until the gate is unlocked.
- After it is unlocked, the FSM enters a second state where it generates an interrupt request to the failed lock processor.
- When the failed lock processor succeeds in locking the gate, the IRQ is automatically negated and the FSM returns to the idle state. However, if the other processor locks the gate again, the FSM returns to the first state, negates the interrupt request, and waits for the gate to be unlocked again.

The notification interrupt request is implemented in a 3-bit, five-state machine, where two specific states are encoded and program-visible as SEMA4_CP0NTF[GN*n*] and SEMA4_CP1NTF[GN*n*].

Offset: SEMA4_BASE + 0x0080 (SEMA4_CP0NTF)
SEMA4_BASE + 0x0088 (SEMA4_CP1NTF)

Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	GN0	GN1	GN2	GN3	GN4	GN5	GN6	GN7	GN8	GN9	GN10	GN11	GN12	GN13	GN14	GN15
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 15-4. Semaphores Processor *n* IRQ Notification (SEMA4_CP{0,1}NTF)

Table 15-4. SEMA4_CP{0,1}NTF Field Descriptions

Field	Description
GN*n*	Gate *n* Notification. This read-only field is a bitmap of the interrupt request notification from a failed attempt to lock gate *n*. 0 No notification interrupt generated. 1 Notification interrupt generated.

15.3.2.4 Semaphores (Secure) Reset Gate *n* (SEMA4_RSTGT)

Although the intent of the hardware gate implementation specifies a protocol where the locking processor must unlock the gate, it is recognized that system operation may require a reset function to re-initialize the state of any gate(s) without requiring a system-level reset.

To support this special gate reset requirement, the semaphores module implements a secure reset mechanism which allows a hardware gate (or all the gates) to be initialized by following a specific dual-write access pattern. Using a technique similar to that required for the servicing of a software

watchdog timer, the secure gate reset requires two consecutive writes with predefined data patterns from the same processor to force the clearing of the specified gate(s). The required access pattern is:

1. A processor performs a 16-bit write to the SEMA4_RSTGT memory location. The most significant byte (SEMA4_RSTGT[RSTGDP]) must be 0xE2; the least significant byte is a “don’t care” for this reference.
2. The same processor then performs a second 16-bit write to the SEMA4_RSTGT location. For this write, the upper byte (SEMA4_RSTGT[RSTGDP]) is the logical complement of the first data pattern (0x1D) and the lower byte (SEMA4_RSTGT[RSTGTN]) specifies the gate(s) to be reset. This gate field can specify a single gate be cleared or that all gates are cleared.
3. Reads of the SEMA4_RSTGT location return information on the 2-bit state machine (SEMA4_RSTGT[RSTGSM]) which implements this function, the bus master performing the reset (SEMA4_RSTGT[RSTGMS]) and the gate number(s) last cleared (SEMA4_RSTGT[RSTGTN]). Reads of the SEMA4_RSTGT register do not affect the secure reset finite state machine in any manner.

Offset: SEMA4_BASE + 0x0100 (SEMA4_RSTGT) Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	RSTGSM		0	RSTGMS			RSTGTN							
W	RSTGDP															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 15-5. Semaphores (Secure) Reset Gate *n* (SEMA4_RSTGT)

Table 15-5. SEMA4_RSTGT Field Descriptions

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>RSTGSM</td><td>Reset Gate Finite State Machine. The reset state machine is maintained in a 2-bit, three-state implementation, defined as:
00 Idle, waiting for the first data pattern write.
01 Waiting for the second data pattern write.
10 The 2-write sequence has completed. Generate the specified gate reset(s). After the reset is performed, this machine returns to the idle (waiting for first data pattern write) state.
11 This state encoding is never used and therefore reserved.
Reads of the SEMA4_RSTGT register return the encoded state machine value. Note the RSTGSM = 0b10 state is valid for a single machine cycle only, so it is impossible for a read to return this value.</td></tr>
<tr><td>RSTGMS</td><td>Reset Gate Bus Master. This 3-bit read-only field records the logical number of the bus master performing the gate reset function. The reset function requires that the two consecutive writes to this register be initiated by the same bus master to succeed. This field is updated each time a write to this register occurs.
<table>
<tr><th>Master</th><th>Master ID</th></tr>
<tr><td>e200z6</td><td>0</td></tr>
<tr><td>e200z0</td><td>1</td></tr>
<tr><td>eDMA</td><td>2</td></tr>
<tr><td>—</td><td>3</td></tr>
<tr><td>FEC</td><td>4</td></tr>
<tr><td>MLB</td><td>5</td></tr>
<tr><td>FlexRay</td><td>6</td></tr>
<tr><td>—</td><td>7</td></tr>
</table></td></tr>
<tr><td>RSTGTN</td><td>Reset Gate Number. This 8-bit field specifies the specific hardware gate to be reset. This field is updated by the second write.
If RSTGTN < 64, then reset the single gate defined by RSTGTN, else reset all the gates. The corresponding secure IRQ notification state machine(s) are also reset.</td></tr>
<tr><td>RSTGDP</td><td>Reset Gate Data Pattern. This write-only field is accessed with the specified data patterns on the two consecutive writes to enable the gate reset mechanism. For the first write, RSTGDP = 0xe2 while the second write requires RSTGDP = 0x1d.</td></tr>
</table>

15.3.2.5 Semaphores (Secure) Reset IRQ Notification (SEMA4_RSTNTF)

As with the case of the secure reset function and the hardware gates, it is recognized that system operation may require a reset function to re-initialize the state of the IRQ notification logic without requiring a system-level reset.

To support this special notification reset requirement, the semaphores module implements a secure reset mechanism which allows an IRQ notification (or all the notifications) to be initialized by following a specific dual-write access pattern. When successful, the specified IRQ notification state machine(s) are reset. Using a technique similar to that required for the servicing of a software watchdog timer, the secure

reset mechanism requires two consecutive writes with predefined data patterns from the same processor to force the clearing of the IRQ notification(s). The required access pattern is:

1. A processor performs a 16-bit write to the SEMA4_RSTNTF memory location. The most significant byte (SEMA4_RSTNTF[RSTNDP]) must be 0x47; the least significant byte is a “don’t care” for this reference.
2. The same processor performs a second 16-bit write to the SEMA4_RSTNTF location. For this write, the upper byte (SEMA4_RSTNTF[RSTNDP]) is the logical complement of the first data pattern (0xb8) and the lower byte (SEMA4_RSTNTF[RSTNTN]) specifies the notification(s) to be reset. This field can specify a single notification be cleared or that all notifications are cleared.
3. Reads of the SEMA4_RSTNTF location return information on the 2-bit state machine (SEMA4_RSTNTF[RSTNSM]) that implements this function, the bus master performing the reset (SEMA4_RSTNTF[RSTNMS]) and the notification number(s) last cleared (SEMA4_RSTNTF[RSTNTN]). Reads of the SEMA4_RSTNTF register do not affect the secure reset finite state machine in any manner.

Offset: SEMA4_BASE + 0x0104 (SEMA4_RSTNTF) Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	RSTNSM		0	RSTNMS			RSTNTN							
W	RSTNDP															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 15-6. Semaphores (Secure) Reset IRQ Notification (SEMA4_RSTNTF)

Table 15-6. SEMA4_RSTGT Field Descriptions

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>RSTNSM</td><td>Reset Notification Finite State Machine. The reset state machine is maintained in a 2-bit, three-state implementation, defined as:
00 Idle, waiting for the first data pattern write.
01 Waiting for the second data pattern write.
10 The two-write sequence has completed. Generate the specified notification reset(s). After the reset is performed, this machine returns to the idle (waiting for first data pattern write) state.
11 This state encoding is never used and therefore reserved.
Reads of the SEMA4_RSTNTF register return the encoded state machine value. Note the RSTNSM = 0b10 state is valid for a single machine cycle only, so it is impossible for a read to return this value.</td></tr>
<tr><td>RSTNMS</td><td>Reset Notification Bus Master. This 3-bit read-only field records the logical number of the bus master performing the notification reset function. The reset function requires that the two consecutive writes to this register be initiated by the same bus master to succeed. This field is updated each time a write to this register occurs.
<table>
<tr><th>Master</th><th>Master ID</th></tr>
<tr><td>e200z6</td><td>0</td></tr>
<tr><td>e200z0</td><td>1</td></tr>
<tr><td>eDMA</td><td>2</td></tr>
<tr><td>—</td><td>3</td></tr>
<tr><td>FEC</td><td>4</td></tr>
<tr><td>MLB</td><td>5</td></tr>
<tr><td>FlexRAY</td><td>6</td></tr>
<tr><td>—</td><td>7</td></tr>
</table></td></tr>
<tr><td>RSTNTN</td><td>Reset Notification Number. This 8-bit field specifies the specific IRQ notification state machine to be reset. This field is updated by the second write.
If RSTNTN < 64, then reset the single IRQ notification machine defined by RSTNTN, else reset all the notifications.</td></tr>
<tr><td>RSTNDP</td><td>Reset Notification Data Pattern. This write-only field is accessed with the specified data patterns on the two consecutive writes to enable the notification reset mechanism. For the first write, RSTNDP = 0x47 while the second write requires RSTNDP = 0xb8.</td></tr>
</table>

15.4 Functional Description

Multi-processor systems require a function that can be used to safely and easily provide a locking mechanism that is then used by system software to control access to shared data structures, shared hardware resources, and etc. These gating mechanisms are used by the software to serialize (and synchronize) writes to shared data and/or resources to prevent race conditions and preserve memory coherency between processes and processors.

For example, if processor X enters a section of code where shared data values are to be updated or read coherently, it must first acquire a semaphore. This locks, or closes, a software gate. After the gate has been locked, a properly architected software system does not allow other processes (or processors) to execute the same code segment or modify the shared data structure protected by the gate, that is, other

processes/processors are locked out. Many software implementations include a spin-wait loop within the lock function until the locking of the gate is accomplished. After the lock has been obtained, processor X continues execution and updates the data values protected by the particular lock. After the updates are complete, processor X unlocks (or opens) the software gate, allowing other processes/processors access to the updated data values.

There are three important rules that must be followed for a correctly implemented system solution:

- All writes to shared data values or shared hardware resources must be protected by a gate variable.
- After a processor locks a gate, accesses to the shared data or resources by other processes/processors must be blocked. This is enforced by software conventions.
- The processor that locks a particular gate is the only processor that can unlock, or open, that gate.

Information in the hardware gate identifying the locking processor can be useful for system-level debugging.

The Hennessy/Patterson text on computer architecture offers this description of software gating:

> "One of the major requirements of a shared-memory architecture multiprocessor is being able to coordinate processes that are working on a common task. Typically, a programmer will use *lock variables* to synchronize the processes.
>
> The difficulty for the architect of a multiprocessor is to provide a mechanism to decide which processor gets the lock and to provide the operation that locks a variable. Arbitration is easy for shared-bus multiprocessors, since the bus is the only path to memory. The processor that gets the bus locks out all the other processors from memory. If the CPU and bus provide an atomic swap operation, programmers can create locks with the proper semantics. The adjective *atomic* is key, for it means that a processor can both read a location **and** set it to the locked value in the same bus operation, preventing any other processor from reading or writing memory." [Hennessy/Patterson, *Computer Architecture: A Quantitative Approach*, ppg. 471-472]

The classic text continues with a description of the steps required to lock/unlock a variable using an atomic swap instruction.

> "Assume that 0 means unlocked and 1 means locked. A processor first reads the lock variable to test its state. A processor keeps reading and testing until the value indicates that the lock is unlocked. The processor then races against all other processes that were similarly "spin waiting" to see who can lock the variable first. All processes use a swap instruction that reads the old value and stores a 1 into the lock variable. The single winner will see the 0, and the losers will see a 1 that was placed there by the winner. (The losers will continue to set the variable to the locked value, but that doesn't matter.) The winning processor executes the code after the lock and then stores a 0 into the lock when it exits, starting the race all over again. Testing the old value and then setting to a new value is why the atomic swap instruction is called *test and set* in some instruction sets." [Hennessy/Patterson, *Computer Architecture: A Quantitative Approach*, ppg. 472-473]

The sole drawback to a hardware-based semaphore module is the limited number of semaphores versus the infinite number that can be supported with Power Architecture reservation instructions.

15.4.1 Semaphore Usage

Example 1: Inter-processor communication done with software interrupts and semaphores...

- The Z0 uses software interrupts to tell the Z6 that new data is available, or the Z6 does the same to tell the Z0 that there is new data available for transmission.
- Because only eight software interrupts are available, the user may need RAM locations or general-purpose registers in the SIU to refine the meaning of the software interrupt.
- Messages are passed between cores in a defined section of system RAM.
- Before a core updates a message, it must check the associated semaphore to see if the other core is in the process of updating the same message. If the RAM not being updated, then the semaphore must first be locked, then the message can be updated. A software interrupt can be sent to the other core and the semaphore can be unlocked. If the RAM is being updated, the CPU must wait for the other core to unlock the semaphore before proceeding with update.
- Using the same memory location for bidirectional communication might be difficult, so two one-way message areas might work better.
  - For example, if both cores want to update the same location, then the following sequence may occur.

    1. The Z0 locks the semaphore, updates the memory, unlocks the semaphore, and generates a software interrupt to the Z6.

    2. Before the Z6 takes the software interrupt request, it finds the semaphore to be unlocked, so it writes new data to the memory.

    3. The Z6 software interrupt ISR reads the data sent to the Z0, not the data sent from the Z0, and performs an incorrect operation.
  - Semaphores do not prevent this situation from occurring.

Example 2: Coherent read done with semaphores...

- The Z6 wants to coherently read a section of shared memory.
- The Z6 should check that the semaphore for the shared memory is not currently set.
- The Z6 should set the semaphore for the shared memory to prevent the Z0 from updating the shared memory.
- The Z6 reads the required data, then unlock the semaphore.

15.5 Initialization Information

The reset state of the semaphores module allows it to begin operation without the need for any further initialization. All the internal state machines are cleared by any reset event, allowing the module to immediately begin operation.

15.6 Application Information

In an operational multi-core system, most interactions involving the semaphores module involves reads and writes to the SEMA4_GATE*n* registers for implementation of the hardware-enforced software gate functions. Typical code segments for gate functions perform the following operations:

- To lock (close) a gate
  - The processor performs a byte write of logical_processor_number + 1 to gate[i]
  - The processor reads back gate[i] and checks for a value of logical_processor_number + 1

    If the compare indicates the expected value

    then the gate is locked; proceed with the protected code segment

    else

    lock operation failed;

    repeat process beginning with byte write to gate[i] in spin-wait loop, or

    proceed with another execution path and wait for failed lock interrupt notification

A simple C-language example of a gatelock function is shown in Example 15-1. This function follows the Hennessy/Patterson example.

Example 15-1. Sample Gatelock Function

```
#define UNLOCK      0
#define CP0_LOCK    1
#define CP2_LOCK    2

void gateLock (n)
int   n;                          /* gate number to lock */
{
  int   i;
  int   current_value;
  int   locked_value;

  i = processor_number();  /* obtain logical CPU number */

  if (i == 0)
    locked_value = CP0_LOCK;
  else
    locked_value = CP1_LOCK;

  /* read the current value of the gate and wait until the state == UNLOCK */
  do {
    current_value = gate[n];
  } while (current_value != UNLOCK);

  /* the current value of the gate == UNLOCK. attempt to lock the gate for this
     processor. spin-wait in this loop until gate ownership is obtained */
  do {
    gate[n] = locked_value;   /* write gate with processor_number + 1 */
    current_value = gate[n];  /* read gate to verify ownership was obtained */
  } while (current_value != locked_value);
}
```

- To unlock (open) a gate
  - After completing the protected code segment, the locking processor performs a byte write of zeroes to gate[i], unlocking (opening) the gate

In this example, a reference to `processor_number()` is used to retrieve this hardware configuration value. Typically, the logical processor numbers are defined by a hardwired input vector to the individual

cores. The exact method for accessing the logical processor number varies by architecture. For Power Architecture cores, there is a processor ID register (PIR) which is SPR 286 and contains this value. A single instruction can be used to move the contents of the PIR into a general-purpose register: mfspr rx,286 where rx is the destination GPRn. Other architectures may support a specific instruction to move the contents of the logical processor number into a general-purpose register, e.g., **`rdcpn rx`** for a read CPU number instruction.

If the optional failed lock IRQ notification mechanisms are used, then accesses to the related registers (SEMA4_CP*n*INE, SEMA4_ CP*n*NTF) are required. There is no required negation of the failed lock write notification interrupt as the request is automatically negated by the semaphores module once the gate has been successfully locked by the failing processor.

Finally, in the event a system state requires a software-controlled reset of a gate or IRQ notification register(s), accesses to the secure reset control registers (SEMA4_RSTGT, SEMA4_RSTNTF) are required. For these situations, it is recommended that the appropriate IRQ notification enable(s) (SEMA4_CP*n*INE) bits be disabled before initiating the secure reset 2-write sequence to avoid any race conditions involving spurious notification interrupt requests.

15.7 DMA Requests

There are no DMA requests associated with the IPS_Semaphore block.

15.8 Interrupt Requests

The semaphore interrupt requests are connected to the interrupt controller as described in Chapter 10, Interrupts and Interrupt Controller (INTC).

Chapter 16
AMBA Crossbar Switch (AXBS)

16.1 Introduction

This chapter describes the multi-port crossbar switch (AXBS), which supports simultaneous connections between six master ports and six slave ports. The AXBS supports a 32-bit address bus width and a 64-bit data bus width at all master and slave ports.

16.1.1 Block Diagram

Figure 16-1 shows a block diagram of the crossbar switch.

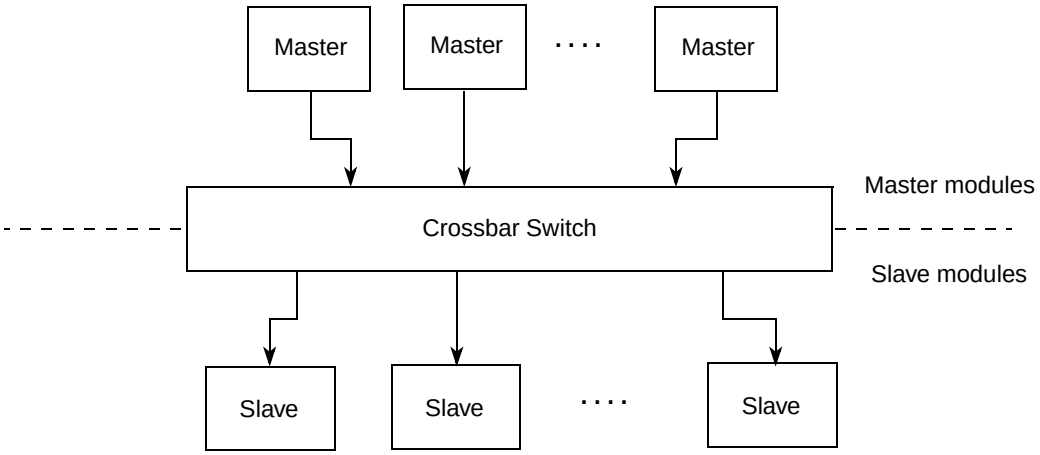


Figure 16-1. AXBS Block Diagram

16.1.2 AXBS Controller Configuration

The AMBA Crossbar Switch (AXBS) supports six masters running at system frequency. The master data bus width is 64 bits. Table 16-1 summarizes the crossbar master port assignments and the Master IDs.

Table 16-1. Master Assignments and Master IDs

AXBS Port	AXBS Module	Master ID
M0	Z6 Core	0
	Z6 Nexus	8
M1	eDMA	2
M2	Off Platform (MLB)	5
M3	FEC	4

Table 16-1. Master Assignments and Master IDs

AXBS Port	AXBS Module	Master ID
M6	Off Platform (FlexRay)	6
M7	Z0 Core	1

The AXBS supports six slaves running at system frequency. The slave data bus width is 64 bits. Table 16-2 summarizes the crossbar slave port assignments.

Table 16-2. Slave Port Assignments

AXBS Port	AXBS Module
S0	Flash port dedicated to Z6
S1	Flash port for all other masters
S2	512K SRAM at address 0x4000_0000
S3	80K SRAM at address 0x4008_0000
S6	AIPS A
S7	AIPS B

16.1.3 Overview

The AXBS allows for concurrent transactions to occur from any master port to any slave port. It is possible for all master ports and slave ports to be in use at the same time as a result of independent master requests. If a slave port is simultaneously requested by more than one master port, arbitration logic selects the higher priority master and grant it ownership of the slave port. All other masters requesting that slave port are stalled until the higher priority master completes its transactions.

By default, requesting masters are granted access based on a fixed priority. A round-robin priority mode also is available. In this mode, requesting masters are treated with equal priority and are granted access to a slave port in round-robin fashion, based on the ID of the last master to be granted access. A block diagram of the AXBS is shown in Figure 16-1.

The AXBS can place a slave port in a low-power park mode to avoid dissipating any power transitional address, control or data signals when the master port is not actively accessing the slave port. There is a one-cycle arbitration overhead for exiting low-power park mode.

16.1.4 Features

- Six master ports:
  - Z6 core/Nexus
  - eDMA
  - Media Local Bus (MLB)
  - Fast Ethernet Controller (FEC)
  - FlexRay
  - Z0 core

- Six slave ports
  - Flash port dedicated to Z6 core
  - Flash port for all other masters (refer to Chapter 12, Flash Memory Array and Control, for information on accessing flash memory)
  - 512K SRAM at address 0x4000_0000
  - 80K SRAM at address 0x4008_0000
  - AIPS A
  - AIPS B
- 32-bit address, 64-bit data paths
- Fully concurrent transfers between independent master and slave ports

16.1.5 Modes of Operation

16.1.5.1 Normal Mode

In normal mode, the AXBS provides the register interface and logic that controls crossbar switch configuration.

16.1.5.2 Debug Mode

The AXBS operation in debug mode is identical to operation in normal mode.

16.2 Memory Map and Register Definition

The memory map for the AXBS program-visible registers is shown in Table 16-3.

Table 16-3. AXBS Register Memory Map

Offset from AXBS_BASE (0xFFF0_4000)	Register	Access	Reset Value	Section/Page
0x0000	XBAR_MPR0—Master Priority Register, Slave Port 0	R/W	0x5400_3210	16.2.1.1/16-4
0x0004–0x000F	Reserved			
0x0010	XBAR_SGPCR0—General-Purpose Control Register, Slave Port 0	R/W	0x0000_0000	16.2.1.2/16-6
0x0014–0x00FF	Reserved			
0x0100	XBAR_MPR1—Master Priority Register, Slave Port 1	R/W	0x5400_3210	16.2.1.1/16-4
0x0104–0x010F	Reserved			
0x0110	XBAR_SGPCR1—General-Purpose Control Register, Slave Port 1	R/W	0x0000_0000	16.2.1.2/16-6
0x0114–0x01FF	Reserved			
0x0200	XBAR_MPR2—Master Priority Register, Slave Port 2	R/W	0x5400_3210	16.2.1.1/16-4

Table 16-3. AXBS Register Memory Map (continued)

Offset from AXBS_BASE (0xFFF0_4000)	Register	Access	Reset Value	Section/Page
0x0204–0x020F	Reserved			
0x0210	XBAR_SGPCR2—General-Purpose Control Register, Slave Port 2	R/W	0x0000_0000	16.2.1.2/16-6
0x0214–0x02FF	Reserved			
0x0300	XBAR_MPR3—Master Priority Register, Slave Port 3	R/W	0x5400_3210	16.2.1.1/16-4
0x0304–0x030F	Reserved			
0x0310	XBAR_SGPCR3—General-Purpose Control Register, Slave Port 3	R/W	0x0000_0000	16.2.1.2/16-6
0x0314–0x05FF	Reserved			
0x0600	XBAR_MPR6—Master Priority Register, Slave Port 6	R/W	0x5400_3210	16.2.1.1/16-4
0x0604–0x060F	Reserved			
0x0610	XBAR_SGPCR6—General-Purpose Control Register, Slave Port 6	R/W	0x0000_0000	16.2.1.2/16-6
0x0614–0x06FF	Reserved			
0x0700	XBAR_MPR7—Master Priority Register, Slave Port 7	R/W	0x5400_3210	16.2.1.1/16-4
0x0704–0x070F	Reserved			
0x0710	XBAR_SGPCR7— General-Purpose Control Register, Slave Port 7	R/W	0x0000_0000	16.2.1.2/16-6
0x0714–0x0EFF	Reserved			
0x0F00	XBAR_MGPCR7—Master General Purpose Register, Master Port 7	R/W	0x0000_0000	16.2.1.3/16-8
0x0F04–0x3FFF	Reserved			

16.2.1 Register Descriptions

There are two registers for each slave port of the AXBS. The registers can only be accessed in supervisor mode using 32-bit accesses.

The slave SGPCR also features a bit (RO), which when written with a 1, prevents all slave registers for that port from being written to again until a reset occurs. The registers remain readable, but future write attempts have no effect on the registers and are terminated with an error response.

16.2.1.1 Master Priority Registers (XBAR_MPR*n*)

The XBAR_MPR for a slave port sets the priority of each master port when operating in fixed priority mode. They are ignored in round-robin priority mode unless more than one master has been assigned high priority by a slave.

NOTE

Masters must be assigned unique priority levels.

The master priority register can only be accessed in supervisor mode with 32-bit accesses. After the read only (RO) bit is set in the slave general-purpose control register, the master priority register can only be read. Attempts to write to it have no effect on the MPR and result in an error.

NOTE

XBAR_MPR must be written with a read/modify/write for code compatibility.

Address: AXBS_BASE + 0x0000 (XBAR_MPR0) AXBS_BASE + 0x0300 (XBAR_MPR3)
AXBS_BASE + 0x0100 (XBAR_MPR1) AXBS_BASE + 0x0600 (XBAR_MPR6)
AXBS_BASE + 0x0200 (XBAR_MPR2) AXBS_BASE + 0x0700 (XBAR_MPR7) Access: Supervisor read/write

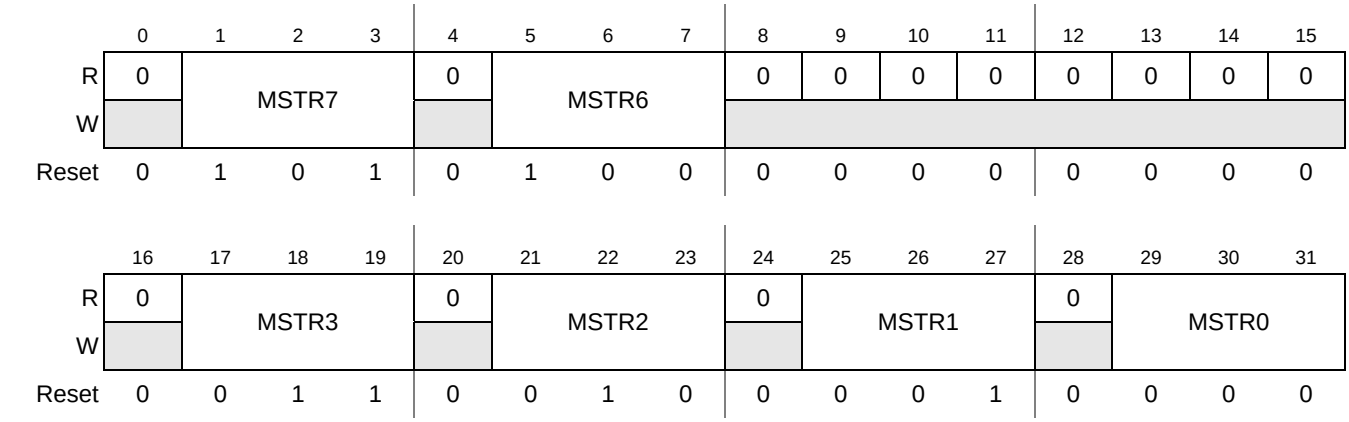


Figure 16-2. Master Priority Registers (XBAR_MPR*n*)

Table 16-4. XBAR_MPR*n* Descriptions

Field	Description
MSTR7	Master 7 priority. Set the arbitration priority for master port 6 on the associated slave port. 000 This master has the highest priority when accessing the slave port. ⋮ 101 This master has the lowest priority when accessing the slave port. 110–111 Invalid values
MSTR6	Master 6 priority. Set the arbitration priority for master port 6 on the associated slave port. 000 This master has the highest priority when accessing the slave port. ⋮ 101 This master has the lowest priority when accessing the slave port. 110–111 Invalid values
MSTR3	Master 3 priority. Set the arbitration priority for master port 3 on the associated slave port. 000 This master has the highest priority when accessing the slave port. ⋮ 101 This master has the lowest priority when accessing the slave port. 110–111 Invalid values

Table 16-4. XBAR_MPR*n* Descriptions (continued)

Field	Description
MSTR2	Master 2 priority. Set the arbitration priority for master port 2 on the associated slave port. 000 This master has the highest priority when accessing the slave port. ⋮ 101 This master has the lowest priority when accessing the slave port. 110–111 Invalid values
MSTR1	Master 1 priority. Set the arbitration priority for master port 1 on the associated slave port. 000 This master has the highest priority when accessing the slave port. ⋮ 101 This master has the lowest priority when accessing the slave port. 110–111 Invalid values
MSTR0	Master 0 priority. Set the arbitration priority for master port 0 on the associated slave port. 000 This master has the highest priority when accessing the slave port. ⋮ 101 This master has the lowest priority when accessing the slave port. 110–111 Invalid values

16.2.1.2 Slave General-Purpose Control Registers (XBAR_SGPCR*n*)

The XBAR_SGPCR*n* of a slave port controls several features of the slave port, including the following:

- Round-robin or fixed arbitration policy for a particular slave port
- Write protection of any slave port registers
- Parking algorithm used for a slave port

The PARK field indicates which master port this slave port parks on when no active access attempts are being made to the slave and the parking control field is set to park on a specific master. XBAR_SGPCR*n*[PARK] must only be programmed to select master ports that are actually available on the device, otherwise undefined behavior results. The low-power park feature can result in an overall power savings if the slave port is not saturated; however, an extra clock of latency results whenever any master tries to access a slave (not being accessed by another master) because it is not parked on any master.

The XBAR_SGPCR can only be accessed in supervisor mode with 32-bit accesses. After the RO (read only) bit is set in the XBAR_SGPCR, the XBAR_SGPCR and the XBAR_MPR can only be read. Attempts to write to them have no effect and results in an error.

NOTE

Some of the unused bits in the SGPCR*n* registers are writeable and readable, but they serve no function. Setting any of these bits has no effect on the operation of this module.

Address: AXBS_BASE + 0x0010 (XBAR_SGPCR0) AXBS_BASE + 0x0310 (XBAR_SGPCR3)
AXBS_BASE + 0x0110 (XBAR_SGPCR1) AXBS_BASE + 0x0610 (XBAR_SGPCR6)
AXBS_BASE + 0x0210 (XBAR_SGPCR2) AXBS_BASE + 0x0710 (XBAR_SGPCR7) Access: Supervisor read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	RO[1]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	ARB		0	0	PCTL		0	PARK		
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

[1] After this bit is set, only a hardware reset clears it.

Figure 16-3. Slave General-Purpose Control Registers (XBAR_SGPCR*n*)

Table 16-5. XBAR_SGPCR*n* Field Descriptions

Field	Description
RO	Read only. Used to force all of a slave port's registers to be read-only. After written to 1, it can only be cleared by hardware reset. 0 All this slave port's registers can be written. 1 All this slave port's registers are read-only and cannot be written (attempted writes have no effect and result in an error response).
ARB	Arbitration mode. Used to select the arbitration policy for the slave port. This field is initialized by hardware reset. 00 Fixed priority using MPR 01 Round-robin priority 10 Invalid value 11 Invalid value
PCTL	Parking control. Used to select the parking algorithm used by the slave port. This field is initialized by hardware reset. 00 When no master is making a request, the arbiter parks the slave port on the master port defined by the PARK control field. 01 POL—Park on last. When no master is making a request, the arbiter parks the slave port on the last master to own the slave port. 10 LPP—Low-power park. When no master is making a request, the arbiter parks the slave port on no master and drives all slave port outputs to a safe state. 11 Invalid value
PARK	Park. Used to determine which master port this slave port parks on when no masters are actively making requests. PCTL must be set to 00. 000 Park on master port 0 001 Park on master port 1 010 Park on master port 2 011 Park on master port 3 100 Invalid value 101 Invalid value 110 Park on master port 6 111 Park on master port 7

16.2.1.3 Master General Purpose Control Registers (XBAR_MGPCR*n*)

The Master General Purpose Control Register (XBAR_MGPCR) controls the arbitration policy during undefined length burst accesses. The AULB (Arbitrate on Undefined Length Bursts) field determines whether or not arbitration occurs for the slave port the master owns when the master is performing undefined length burst accesses.

The MGPCR can only be accessed in supervisor mode with 32-bit access.

Address AXBS_BASE + 0x0F00 (XBAR_MGPCR7) Access: Supervisor read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	AULB		
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 16-4. Master General-Purpose Control Registers (XBAR_MGPCR*n*)

Table 16-6. XBAR_MGPCR*n* Field Descriptions

Field	Description
AULB	Arbitration on Undefined Length Bursts. This field is used to select the arbitration policy during undefined length bursts by this master. This field is cleared by hardware reset. 000 No arbitration during undefined length bursts 001 Arbitration allowed on every beat of an undefined length burst 010 Arbitration allowed after four beats of an undefined length burst 011 Arbitration allowed after eight beats of an undefined length burst 100 Arbitration allowed after sixteen beats of an undefined length burst 101–111 Reserved

16.3 Functional Description

This section describes the functionality of the AXBS in more detail.

16.3.1 Overview

The main goal of the AXBS is to increase overall system performance by allowing multiple masters to communicate concurrently with multiple slaves. To maximize data throughput, it is essential to keep arbitration delays to a minimum.

This section examines data throughput from the point of view of masters and slaves, detailing when the AXBS stalls masters, or inserts bubbles on the slave side.

16.3.2 General Operation

When a master makes an access to the AXBS from an idle master state, the access is taken immediately by the AXBS. If the targeted slave port of the access is available (that is, the requesting master is currently granted ownership of the slave port), the access is immediately presented on the slave port. It is possible to make single clock (zero wait state) accesses through the AXBS by a granted master. If the targeted slave port of the access is busy or parked on a different master port, the requesting master receives wait states until the targeted slave port can service the master request. The latency in servicing the request depends on each master's priority level and the responding slave's access time.

Because the AXBS appears to be simply another slave to the master device, the master device has no indication that it owns the slave port it is targeting. While the master does not have control of the slave port it is targeting, it is wait-stated.

A master is given control of a targeted slave port only after a previous access to a different slave port has completed, regardless of its priority on the newly targeted slave port. This prevents deadlock from occurring when a master has the following conditions:

- Outstanding request to slave port A that has a long response time
- Pending access to a different slave port B
- Lower priority master also makes a request to the different slave port B.

In this case, the lower priority master is granted bus ownership of slave port B after a cycle of arbitration, assuming the higher priority master slave port A access is not terminated.

After a master has control of the slave port it is targeting, the master remains in control of that slave port until it gives up the slave port by running an IDLE cycle, leaves that slave port for its next access, or loses control of the slave port to a higher priority master with a request to the same slave port. However, because all masters run a fixed-length burst transfer to a slave port, it retains control of the slave port until that transfer sequence is completed. In round-robin arbitration mode, the current master is forced to hand off bus ownership to an alternately requesting master at the end of its current transfer sequence.

When a slave bus is idled by the AXBS, it can be parked on the master port using the PARK bits in the XBAR_SGPCR (slave general-purpose control register), or on the last master to have control of the slave port. This can avoid the initial clock of the arbitration delay if the master must arbitrate to gain control of the slave port. The slave port can also be put into low-power park mode to save power.

16.3.3 Master Ports

The AXBS terminates an access and it is not allowed to pass through the AXBS unless the master currently is granted access to the slave port to which the access is targeted. A master access is taken if the slave port to which the access decodes is either currently servicing the master or is parked on the master. In this case, the AXBS is completely transparent and the master access is immediately transmitted on the slave bus and no arbitration delays are incurred. A master access stall if the access decodes to a slave port that is busy serving another master, parked on another master or is in low-power park mode.

If the slave port is currently parked on another master or is in low-power park mode, and no other master is requesting access to the slave port, then only one clock of arbitration is incurred. If the slave port is currently serving another master of a lower priority and the master has a higher priority than all other

requesting masters, then the master gains control over the slave port as soon as the data phase of the current access is completed. If the slave port is currently servicing another master of a higher priority, then the master gains control of the slave port after the other master releases control of the slave port if no other higher priority master is also waiting for the slave port.

A master access is responded to with an error if the access decodes to a location not occupied by a slave port. This is the only time the AXBS directly responds with an error response. All other error responses received by the master are the result of error responses on the slave ports being passed through the AXBS.

16.3.4 Slave Ports

The goal of the AXBS with respect to the slave ports is to keep them 100% saturated when masters are actively making requests. To do this the AXBS must not insert any bubbles onto the slave bus unless absolutely necessary.

There is only one instance when the AXBS forces a bubble onto the slave bus when a master is actively making a request. This occurs when a handoff of bus ownership occurs and there are no wait states from the slave port. A requesting master which does not own the slave port is granted access after a one clock delay.

The only other time the AXBS has control of the slave port is when no masters are making access requests to the slave port and the AXBS is forced to either park the slave port on a specific master, or place the slave port into low-power park mode. In these cases, the AXBS forces IDLE for the transfer type.

16.3.5 Priority Assignment

Each master port must be assigned a unique 2-bit priority level in fixed priority mode. If multiple master ports are assigned the same priority level within a register (XBAR_MPR) undefined behavior results.

16.3.6 Arbitration

The AXBS supports two arbitration schemes; a simple fixed-priority comparison algorithm, and a round-robin fairness algorithm. The arbitration scheme is independently programmable for each slave port.

16.3.6.1 Fixed Priority Operation

When operating in fixed-priority arbitration mode, each master is assigned a unique priority level in the XBAR_MPR. If two masters both request access to a slave port, the master with the highest priority in the selected priority register gains control over the slave port.

Any time a master makes a request to a slave port, the slave port checks to see if the new requesting master’s priority level is higher than that of the master that currently has control over the slave port (if any). The slave port does an arbitration check at every clock edge to ensure that the proper master (if any) has control of the slave port.

If the new requesting master's priority level is higher than that of the master that currently has control of the slave port, the higher priority master is granted control at the termination of any currently pending access, assuming the pending transfer is not part of a burst transfer.

A new requesting master must wait until the end of the fixed-length burst transfer, before it is granted control of the slave port. But if the new requesting master's priority level is lower than that of the master that currently has control of the slave port, the new requesting master is forced to wait until the master that currently has control of the slave port is finished accessing the current slave port.

16.3.6.2 Round-Robin Priority Operation

When operating in round-robin mode, each master is assigned a relative priority based on the master port number. This relative priority is compared to the port number of the last master to perform a transfer on the slave bus. The highest priority requesting master becomes the owner of the slave bus at the next transfer boundary (accounting for fixed-length burst transfers). Priority is based on how far ahead the port number of the requesting master is to the port number of the last master.

After granted access to a slave port, a master may perform as many transfers as desired to that port until another master makes a request to the same slave port. The next master in line is granted access to the slave port when the current transfer is completed, or possibly on the next clock cycle if the current master has no pending access request.

As an example of arbitration in round-robin mode, assume the three masters have ID's 0, 1, and 2. If the last master of the slave port was master 1, and masters 0 and 2 make simultaneous requests, they are serviced in the order 2 and then 0 assuming no further requests are made.

As another example, if master 1 is waiting on a response from a slow slave and has no further pending access to that slave, no other masters are requesting, and master 0 then makes a request, master 0's request is granted on the next clock (assuming that master 1's transfer is not a burst transfer), and the request information for master 0 is driven to the slave as a pending access. If master 2 were to make a request after master 0 has been granted access, but prior to master 0's access being accepted by the slave, master 0 maintains the grant on the slave port, and master 2 is delayed until the next arbitration boundary, which occurs after the transfer is complete. The round-robin pointer is reset to 0, so if master 1 has another request that occurs before master 0's transfer completes, master 1 is the granted the bus. This implies a worst case latency of *N* transfers for a system with *N* masters.

Parking may continue to be used in round-robin mode, but affects the round-robin pointer unless the parked master actually performs a transfer. Handoff to the next master in line occurs after one cycle of arbitration.

The slave port does an arbitration check at every clock edge to ensure that the proper master (if any) has control of the slave port.

A new requesting master must wait until the end of the fixed-length burst transfer, before it is granted control of the slave port. If the new requesting master's priority level is lower than that of the master that currently has control of the slave port, the new requesting master is forced to wait until the master that currently has control of the slave port completes its access.

16.3.6.2.1 Parking

If no master is currently requesting the slave port, the slave port is parked. The slave port parks in one of three places, indicated by the value of the PCTL field in the XBAR_SGPCR.

- If park-on-specific master mode is selected, the slave port parks on the master designated by the PARK field. When the master accesses the slave port again, a one clock arbitration penalty is incurred only for an access request made by another master port to the slave port. No other arbitration penalties are incurred. All other masters pay a one clock penalty.
- If park-on-last (POL) mode is selected, then the slave port parks on the last master to access it, passing that master's signals through to the slave bus. When the master accesses the slave port again, no other arbitration penalties are incurred except that a one clock arbitration penalty is incurred for each access request to the slave port made by another master port. All other masters pay a one clock penalty.
- If the low-power-park (LPP) mode is selected, then the slave port enters low-power park mode. It is not under control by any master and does not transmit any master signals to the slave bus. All slave bus activity halts because all slave bus signals are not toggling. This saves power if the slave port is not used for some time. However, when a master does make a request to a slave port parked in low-power-park, a one clock arbitration delay is incurred to get ownership of the slave port.

Chapter 17
Peripheral Bridge (AIPS-lite)

17.1 Introduction

The AIPS-lite acts as an interface between the system bus and lower bandwidth peripherals.

17.1.1 Block Diagram

A simplified block diagram of the AIPS-lite illustrates the functionality and interdependence of major blocks (see Figure 17-1).

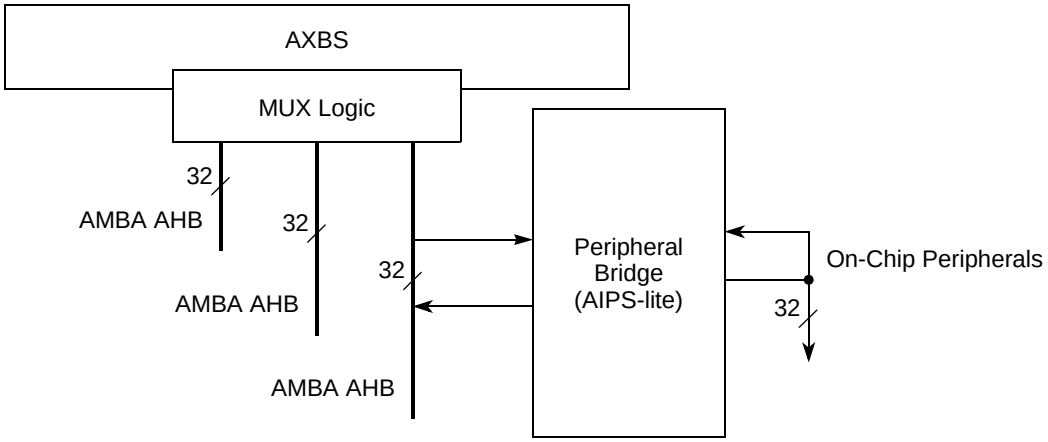


Figure 17-1. AIPS-lite Block Diagram

17.1.2 Features

The AIPS-lite has these major features:

- AIPS-lite supports the IPS slave interface signals. This interface is meant for slave peripherals only.
- AIPS-lite supports 32-bit IPS peripherals. (Byte, halfword, and word reads and write are supported to each.)
- Read and write accesses of 32 bits or less require two clocks, provided they do not cross a 32-bit boundary.
  - Read and write accesses that cross a 32-bit boundary are not supported.
- The peripherals connected to the AIPS-lite may be configured in groups to run at less than the system clock frequency. See Section 5.3, Clock Dividers, in Chapter 5, System Clock Description, for a description of these groups.

17.1.3 Modes of Operation

The AIPS-lite has only one operating mode.

17.2 External Signal Description

The AIPS-lite has no external signals.

17.3 Memory Map and Register Description

The AIPS-lite does not contain any user-programmable registers.

17.4 Functional Description

The AIPS-lite serves as an interface between an AHB 2.v6 system bus and the peripheral interface bus. It functions as a protocol translator.

Accesses that fall within the address space of the AIPS-lite are decoded to provide individual module selects for peripheral devices on the peripheral bus interface.

See the peripherals section of Table 2-1 for a description of which peripherals are allocated to which 16 KB memory space in the AIPS-lite address map.

17.4.1 Read Cycles

Two-clock read accesses are possible with the AIPS-Lite when the reference size is 32 bits or smaller. This module does not support any type of misaligned read accesses crossing a 32-bit boundary.

17.4.2 Write Cycles

Two-clock write accesses are possible with the AIPS-Lite when the reference size is 32 bits or smaller. This module does not support any type of misaligned write accesses crossing a 32-bit boundary.

Chapter 18
Memory Protection Unit (MPU)

18.1 Introduction

The memory protection unit (MPU) provides hardware access control for all memory references generated in a device. Using pre-programmed region descriptors that define memory spaces and their associated access rights, the MPU concurrently monitors all system bus transactions and evaluates the appropriateness of each transfer. Memory references with sufficient access control rights are allowed to complete, but references that are not mapped to any region descriptor or have insufficient rights are terminated with a protection error response.

The MPU implements a set of program-visible region descriptors that monitor all system bus addresses. The result is a hardware structure with a two-dimensional connection matrix, where the region descriptors represent one dimension and the individual system bus addresses and attributes are the second dimension.

NOTE

The MPU module is not implemented on the PXN20.

18.1.1 Block Diagram

A simplified block diagram illustrates how the MPU block is connected to the four AXBS-lite MPU ports and the shared port splitter (see Figure 18-1).

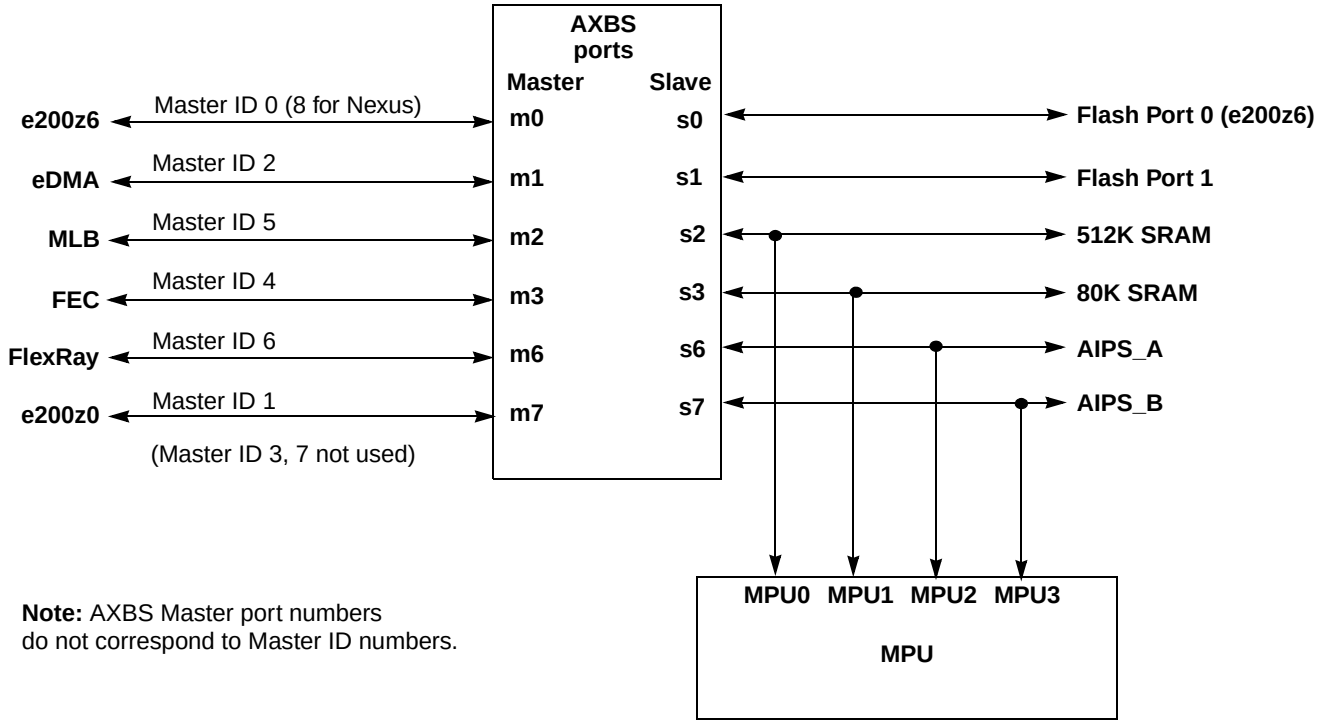


Figure 18-1. MPU Connections to AXBS-lite

Table 18-1. Master Assignments and Master IDs

AXBS Port	AXBS Module	Master ID
M0	Z6 Core	0
	Z6 Nexus	8
M1	eDMA	2
M2	Off Platform (MLB)	5
M3	FEC	4
M6	Off Platform (FlexRay)	6
M7	Z0 Core	1

Figure 18-2.

18.1.2 Features

The MPU has these major features:

Support for 16 memory region descriptors, each 128 bits in size

- Specification of start and end addresses provide granularity for region sizes from 32 bytes to 4 GB
- MPU is invalid at reset, thus no access restrictions are enforced

  - Two types of access control definitions: two processor core bus masters (e200z6 and e200z0) support the traditional {read, write, execute} permissions with independent definitions for supervisor and user mode accesses; the remaining three non-core bus masters (DMA, FlexRay, and AIPS) support {read, write} attributes
  - Automatic hardware maintenance of the region descriptor valid bit removes issues associated with maintaining a coherent image of the descriptor
  - Alternate memory view of the access control word for each descriptor provides an efficient mechanism to dynamically alter the access rights of a descriptor only
  - For overlapping region descriptors, priority is given to permission granting over access denying as this approach provides more flexibility to system software
- Support for four AHB MPU port connections
  - AIPS_A, AIPS_B, SRAM_A, SRAM_B
  - MPU hardware monitors every AHB MPU port access using the pre-programmed memory region descriptors
  - An access protection error is detected if a memory reference does not hit in any memory region or the reference is flagged as illegal in all memory regions where it does hit; in the event of an access error, the AHB reference is terminated with an error response and the MPU inhibits the bus cycle being sent to the targeted slave device
  - 64-bit error registers, one for each AHB MPU port, capture the last faulting address, attributes, and detail information

18.1.3 Modes of Operation

The MPU does not support any special modes of operation.

18.2 Signal Description

The MPU does not include any external signals.

18.3 Memory Map and Registers

This section provides a detailed description of all MPU registers.

18.3.1 Module Memory Map

The MPU memory map is shown in Table 18-2. The address of each register is given as an offset to the MPU base address. Registers are listed in address order, identified by complete name and mnemonic, and list the type of accesses allowed.

The MPU registers can be referenced using 32-bit (word) accesses only. Attempted references using different access sizes, to undefined (reserved) addresses, or with a non-supported access type (for example, a write to a read-only register or a read of a write-only register) generate an error termination.

Table 18-2. MPU Memory Map

Offset from MPU_BASE (0xFFF1_4000)	Register	Access	Reset Value	Section/Page
0x0000	MPU_CESR—MPU control/error status register	R/W	0x0080_4200	18.3.2.1/18-5
0x0004–0x000F	Reserved			
0x0010	MPU_EAR0—MPU error address register, MPU port 0	RO	—[1]	18.3.2.2/18-6
0x0014	MPU_EDR0—MPU error detail register, MPU port 0	RO	—[1]	18.3.2.3/18-7
0x0018	MPU_EAR1—MPU error address register, MPU port 1	RO	—[1]	18.3.2.2/18-6
0x001C	MPU_EDR1—MPU error detail register, MPU port 1	RO	—[1]	18.3.2.3/18-7
0x0020	MPU_EAR2—MPU error address register, MPU port 2	RO	—[1]	18.3.2.2/18-6
0x0024	MPU_EDR2—MPU error detail register, MPU port 2	RO	—[1]	18.3.2.3/18-7
0x0028	MPU_EAR3— MPU error address register, MPU port 3	RO	—[1]	18.3.2.3/18-7
0x002C	MPU_EDR3—MPU error detail register, MPU port 3	RO	—[1]	18.3.2.3/18-7
0x0030–0x03FF	Reserved			
0x0400	MPU_RGD0—MPU region descriptor 0	R/W	—[1]	18.3.2.4/18-8
0x0410	MPU_RGD1—MPU region descriptor 1	R/W	—[1]	18.3.2.4/18-8
0x0420	MPU_RGD2—MPU region descriptor 2	R/W	—[1]	18.3.2.4/18-8
0x0430	MPU_RGD3—MPU region descriptor 3	R/W	—[1]	18.3.2.4/18-8
0x0440	MPU_RGD4—MPU region descriptor 4	R/W	—[1]	18.3.2.4/18-8
0x0450	MPU_RGD5—MPU region descriptor 5	R/W	—[1]	18.3.2.4/18-8
0x0460	MPU_RGD6—MPU region descriptor 6	R/W	—[1]	18.3.2.4/18-8
0x0470	MPU_RGD7—MPU region descriptor 7	R/W	—[1]	18.3.2.4/18-8
0x0480	MPU_RGD8—MPU region descriptor 8	R/W	—[1]	18.3.2.4/18-8
0x0490	MPU_RGD9—MPU region descriptor 9	R/W	—[1]	18.3.2.4/18-8
0x04A0	MPU_RGD10—MPU region descriptor 10	R/W	—[1]	18.3.2.4/18-8
0x04B0	MPU_RGD11—MPU region descriptor 11	R/W	—[1]	18.3.2.4/18-8
0x04C0	MPU_RGD12—MPU region descriptor 12	R/W	—[1]	18.3.2.4/18-8
0x04D0	MPU_RGD13—MPU region descriptor 13	R/W	—[1]	18.3.2.4/18-8
0x04E0	MPU_RGD14—MPU region descriptor 14	R/W	—[1]	18.3.2.4/18-8
0x04F0	MPU_RGD15—MPU region descriptor 15	R/W	—[1]	18.3.2.4/18-8
0x00500–0x07FF	Reserved			
0x0800	MPU_RGDAAC0—MPU RGD alternate access control 0	W	—[1]	18.3.2.5/18-13
0x0804	MPU_RGDAAC1—MPU RGD alternate access control 1	W	—[1]	18.3.2.5/18-13
0x0808	MPU_RGDAAC2—MPU RGD alternate access control 2	W	—[1]	18.3.2.5/18-13

Table 18-2. MPU Memory Map (continued)

Offset from MPU_BASE (0xFFF1_4000)	Register	Access	Reset Value	Section/Page
0x080C	MPU_RGDAAC3—MPU RGD alternate access control 3	W	—[1]	18.3.2.5/18-13
0x0810	MPU_RGDAAC4—MPU RGD alternate access control 4	W	—[1]	18.3.2.5/18-13
0x0814	MPU_RGDAAC5—MPU RGD alternate access control 5	W	—[1]	18.3.2.5/18-13
0x0818	MPU_RGDAAC6—MPU RGD alternate access control 6	W	—[1]	18.3.2.5/18-13
0x081C	MPU_RGDAAC7—MPU RGD alternate access control 7	W	—[1]	18.3.2.5/18-13
0x0820	MPU_RGDAAC8—MPU RGD alternate access control 8	W	—[1]	18.3.2.5/18-13
0x0824	MPU_RGDAAC9—MPU RGD alternate access control 9	W	—[1]	18.3.2.5/18-13
0x0828	MPU_RGDAAC10—MPU RGD alternate access control 10	W	—[1]	18.3.2.5/18-13
0x082C	MPU_RGDAAC11—MPU RGD alternate access control 11	W	—[1]	18.3.2.5/18-13
0x0830	MPU_RGDAAC12—MPU RGD alternate access control 12	W	—[1]	18.3.2.5/18-13
0x0834	MPU_RGDAAC13—MPU RGD alternate access control 13	W	—[1]	18.3.2.5/18-13
0x0838	MPU_RGDAAC14—MPU RGD alternate access control 14	W	—[1]	18.3.2.5/18-13
0x083C	MPU_RGDAAC15—MPU RGD alternate access control 15	W	—[1]	18.3.2.5/18-13
0x0840–0x08FF	Reserved			

[1] See register definition.

18.3.2 Register Descriptions

This section lists the MPU registers in address order and describes the registers and their bit fields.

18.3.2.1 MPU Control/Error Status Register (MPU_CESR)

The MPU_CESR provides one byte of error status and three bytes of configuration information. A global MPU enable/disable bit is also included in this register.

Offset: MPU_BASE + 0x0000 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	MPERR[1]								1	0	0	0	HRL			
W																
Reset	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	NSP				NRGD				0	0	0	0	0	0	0	VLD
W																
Reset	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0

Figure 18-3. MPU Control/Error Status Register (MPU_CESR)

[1] Each MPERR bit can be cleared by writing a one to the bit location.

Table 18-3. MPU_CESR Field Descriptions

Field	Description
MPERR	MPU Port *n* Error, where the MPU port number matches the bit number. Each bit in this read-only field represents a flag maintained by the MPU for signaling the presence of a captured error contained in the MPU_EAR*n* and MPU_EDR*n* registers. The individual bit is set when the hardware detects an error and records the faulting address and attributes. It is cleared when the corresponding bit is written to a logical one. If another error is captured at the exact same cycle as a write of a logical one, this flag remains set. A find-first-one instruction (or equivalent) can be used to detect the presence of a captured error. 0 The corresponding MPU_EAR*n*/MPU_EDR*n* registers do not contain an unread captured error 1 The corresponding MPU_EAR*n*/MPU_EDR*n* registers do contain an unread captured error **Note:** Bit 0 indicates a 512 KB RAM access protection error, bit 1 represents an 80 KB RAM access protection error, bit 2 represents an AIPS_A access protection error, and bit 3 represents an AIPS_B access protection error.
HRL	Hardware Revision Level. This 4-bit read-only field specifies the MPU's hardware and definition revision level. It can be read by software to determine the functional definition of the module. This field reads as 0 on PXN20 .
NSP	Number of MPU Ports. This 4-bit read-only field specifies the number of MPU ports [1–8] connected to the MPU. This field reads as 0b0011 on PXN20.
NRGD	Number of Region Descriptors. This 4-bit read-only field specifies the number of region descriptors implemented in the MPU. The defined encodings include: 0000 8 region descriptors. 0010 16 region descriptors. This field reads as 0b0010 on PXN20.
VLD	Valid. This bit provides a global enable/disable for the MPU. 0 The MPU is disabled. 1 The MPU is enabled. While the MPU is disabled, all accesses from all bus masters are allowed.

18.3.2.2 MPU Error Address Register, MPU Port 0 to 3 (MPU_EAR*n*)

When the MPU detects an access error on MPU port *n*, the 32-bit reference address is captured in this read-only register and the corresponding bit in the MPU_CESR[MPERR] field set. Additional information about the faulting access is captured in the corresponding MPU_EDR*n* register at the same time.

Offset: MPU_BASE + 0x0010 (MPU_EAR0) Access: User read-only
MPU_BASE + 0x0018 (MPU_EAR1)
MPU_BASE + 0x0020 (MPU_EAR2)
MPU_BASE + 0x0028 (MPU_EAR3)

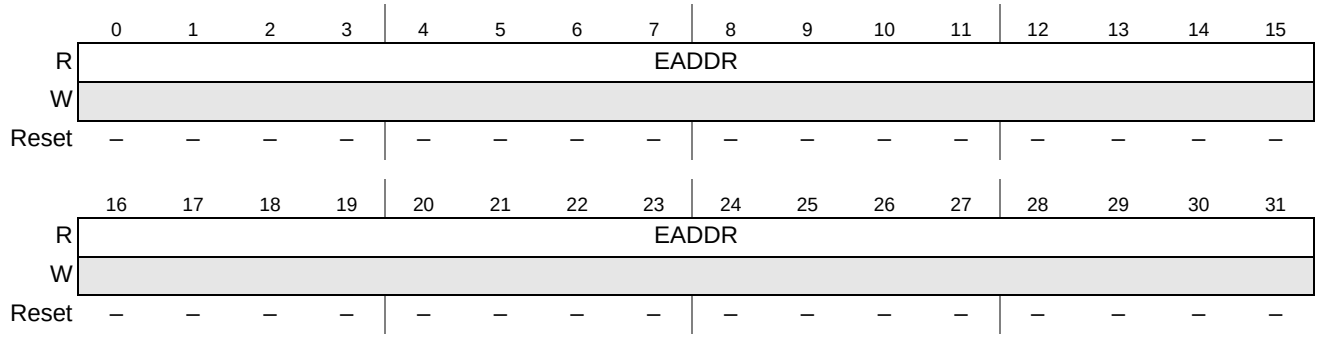


Figure 18-4. MPU Error Address Register, MPU Port *n* (MPU_EAR*n*)

Table 18-4. MPU_EAR*n* Field Descriptions

Field	Description
EADDR	Error Address. This read-only field is the reference address from MPU port *n* that generated the access error.

18.3.2.3 MPU Error Detail Register, MPU Port 0 to 3 (MPU_EDR*n*)

When the MPU detects an access error on MPU port *n*, 32 bits of error detail are captured in this read-only register and the corresponding bit in the MPU_CESR[MPERR] field set. Information on the faulting address is captured in the corresponding MPU_EAR*n* register at the same time.

Offset: MPU_BASE + 0x00014 (MPU_EDR0) Access: User read-only
MPU_BASE + 0x001C (MPU_EDR1)
MPU_BASE + 0x0024 (MPU_EDR2)
MPU_BASE + 0x002C (MPU_EDR3)

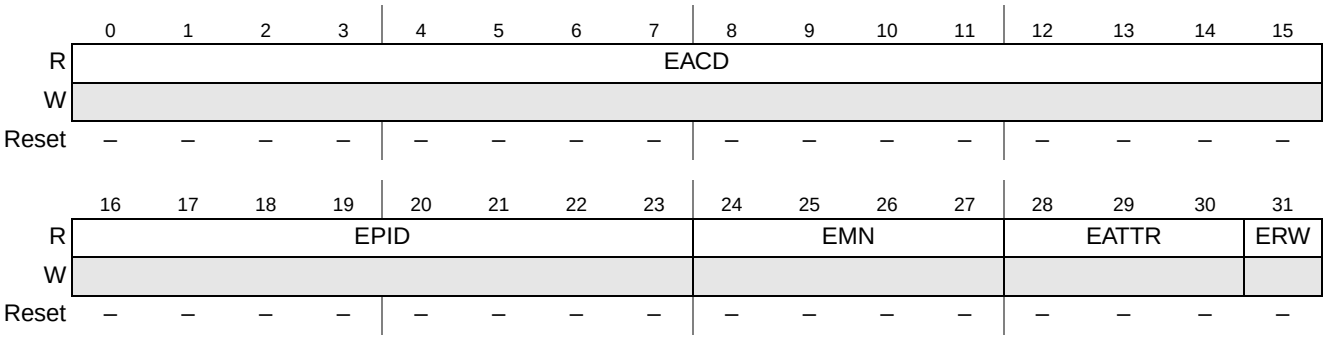


Figure 18-5. MPU Error Detail Register, MPU Port 0 to 3 (MPU_EDR*n*)

Table 18-5. MPU_EDR*n* Field Descriptions

Field	Description
EACD	Error Access Control Detail. This 16-bit read-only field implements one bit per region descriptor and is an indication of the region descriptor hit logically-ANDed with the access error indication. The MPU performs a reference-by-reference evaluation to determine the presence/absence of an access error. When an error is detected, the hit-qualified access control vector is captured in this field. If the MPU_EDR*n* register contains a captured error and the EACD field is all zeroes, this signals an access that did not hit in any region descriptor. All non-zero EACD values signal references that hit in a region descriptor(s), but failed due to a protection error as defined by the specific set bits.
EPID	Error Process Identification. This 8-bit read-only field records the process identifier of the faulting reference. The process identifier is typically driven by processor cores only; for other bus masters, this field is cleared.
EMN	Error Master Number. This 4-bit read-only field records the logical master number of the faulting reference. This field is used to determine the bus master that generated the access error.
EATTR	Error Attributes. This 3-bit read-only field records attribute information about the faulting reference. The supported encodings are defined as: 000 User mode, instruction access. 001 User mode, data access. 010 Supervisor mode, instruction access. 011 Supervisor mode, data access. All other encodings are reserved. For non-core bus masters, the access attribute information is typically wired to supervisor, data (0b011).
ERW	Error Read/Write. This 1-bit read-only field signals the access type (read, write) of the faulting reference. 0 Read. 1 Write.

18.3.2.4 MPU Region Descriptor *n* (MPU_RGD*n*)

Each 128-bit (16 byte) region descriptor specifies a given memory space and the access attributes associated with that space. The descriptor definition is fundamental to the operation of the MPU.

The region descriptors are organized sequentially in the MPU's programming model and each of the four 32-bit words are detailed in the subsequent sections.

18.3.2.4.1 MPU Region Descriptor *n*, Word 0 (MPU_RGD*n*.Word0)

The first word of the MPU region descriptor defines the 0-modulo-32 byte start address of the memory region. Writes to this word clear the region descriptor's valid bit.

Offset: MPU_BASE + 0x400 + (16*n) + 0x0 (MPU_RGDn.Word0) Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	SRTADDR															
W																
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	SRTADDR											0	0	0	0	0
W																
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Figure 18-6. MPU Region Descriptor, Word 0 Register (MPU_RGDn.Word0)

Table 18-6. MPU_RGDn.Word0 Field Descriptions

Field	Description
SRTADDR	Start Address. This field defines the most significant bits of the 0-modulo-32 byte start address of the memory region.

18.3.2.4.2 MPU Region Descriptor *n*, Word 1 (MPU_RGD*n*.Word1)

The second word of the MPU region descriptor defines the 31-modulo-32 byte end address of the memory region. Writes to this word clear the region descriptor's valid bit.

Offset: MPU_BASE + 0x400 + (16*n) + 0x4 (MPU_RGDn.Word1) Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	ENDADDR															
W																
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	ENDADDR											1	1	1	1	1
W																
Reset	–	–	–	–	–	–	–	–	–	–	–	1	1	1	1	1

Figure 18-7. MPU Region Descriptor, Word 1 Register (MPU_RGDn.Word1)

Table 18-7. MPU_RGDn.Word1 Field Descriptions

Field	Description
ENDADDR	End Address. This field defines the most significant bits of the 31-modulo-32 byte end address of the memory region. There are no hardware checks to verify that ENDADDR ≥ SRTADDR; the software must properly load these region descriptor fields.

18.3.2.4.3 MPU Region Descriptor *n*, Word 2 (MPU_RGD*n*.Word2)

The third word of the MPU region descriptor defines the access control rights of the memory region. The access control privileges are dependent on two broad classifications of bus masters. Bus masters 0–3 are

typically reserved for processor cores. The corresponding access control is a 6-bit field defining separate privilege rights for user and supervisor mode accesses as well as the optional inclusion of a process identification field within the definition. Bus masters 4–7 are typically reserved for data movement engines and their capabilities are limited to separate read and write permissions. For these fields, the bus master number refers to the logical master number defined as the AHB `hmaster[3:0]`signal.

For the processor privilege rights, there are three flags associated with this function: {read, write, execute}. In this context, these flags follow the traditional definition:

- Read (r) permission refers to the ability to access the referenced memory address using an operand (data) fetch.
- Write (w) permission refers to the ability to update the referenced memory address using a store (data) instruction.
- Execute (x) permission refers to the ability to read the referenced memory address using an instruction fetch.

The evaluation logic defines the processor access type based on multiple AHB signals: read or write as specified by the `hwrite` signal and the low-order two bits of `hprot[1:0]`, which identify a data reference versus an instruction fetch and the operating mode (supervisor, user) of the requesting processor.

For non-processor data movement engines (bus masters 4–7), the evaluation logic simply uses `hwrite` to determine if the access is a read or write. The `hprot[1:0]` signal is ignored for these masters.

Writes to this word clear the region descriptor's valid bit. Because it is also expected that system software may adjust only the access controls within a region descriptor (MPU_RGD*n*.Word2) as different tasks execute, an alternate programming view of this 32-bit entity is provided. If only the access controls are being updated, this operation should be performed by writing to MPU_RGDAAC*n* (alternate access control *n*) as stores to these locations do not affect the descriptor's valid bit.

Offset: MPU_BASE + 0x400 + (16*n) + 0x8 (MPU_RGDn.Word2) Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	M6RE	M6WE	M5RE	M5WE	M4RE	M4WE	0	0	0	0	0	0	M2PE	M2SM
W																
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	M2SM	M2UM			M1PE	M1SM		M1UM			M0PE	M0SM		M0UM		
W		r	w	x				r	w	x				r	w	x
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Note: Refer to Figure 18-1 to see the Master ID assignments.

Figure 18-8. MPU Region Descriptor, Word 2 Register (MPU_RGD*n*.Word2)

Table 18-8. MPU_RGD*n*.Word2 Field Descriptions

Field	Description
M6RE	Bus Master ID 6 Read Enable. If set, this flag allows bus master ID 6 (FlexRay) to perform read operations. If cleared, any attempted read by bus master ID 6 terminates with an access error and the read is not performed.

Table 18-8. MPU_RGD*n*.Word2 Field Descriptions (continued)

Field	Description
M6WE	Bus Master ID 6 Write Enable. If set, this flag allows bus master ID 6 (FlexRay) to perform write operations. If cleared, any attempted write by bus master ID 6 terminates with an access error and the write is not performed.
M5RE	Bus Master ID 5 Read Enable. If set, this flag allows bus master ID (Media Local Bus) 5 to perform read operations. If cleared, any attempted read by bus master ID 5 terminates with an access error and the read is not performed.
M5WE	Bus Master ID 5 Write Enable. If set, this flag allows bus master ID 5 (Media Local Bus) to perform write operations. If cleared, any attempted write by bus master ID 5 terminates with an access error and the write is not performed.
M4RE	Bus Master ID 4 Read Enable. If set, this flag allows bus master ID (FEC) 4 to perform read operations. If cleared, any attempted read by bus master ID 4 terminates with an access error and the read is not performed.
M4WE	Bus Master ID 4 Write Enable. If set, this flag allows bus master ID (FEC) 4 to perform write operations. If cleared, any attempted write by bus master ID 4 terminates with an access error and the write is not performed.
M2PE	This bit can be read and written to either a 0 or 1, but the MPU behaves as if this bit was permanently tied to 0, so that the PID is not part of the region hit evaluation.
M2SM	Bus Master ID 2 Supervisor Mode Access Control. This 2-bit field defines the access controls for bus master ID 2 (eDMA) when operating in supervisor mode. The M2SM field is defined as: 00 `r, w, x` = read, write and execute allowed. 01 `r, –, x` = read and execute allowed, but no write. 10 `r, w, –` = read and write allowed, but no execute. 11 Same access controls as that defined by M2UM for user mode.
M2UM	Bus Master ID 2 User Mode Access Control. This 3-bit field defines the access controls for bus master ID 2 (eDMA) when operating in user mode. The M2UM field consists of three independent bits, enabling read, write, and execute permissions: `{r,w,x}`. If set, the bit allows the given access type to occur; if cleared, an attempted access of that mode may be terminated with an access error (if not allowed by any other descriptor) and the access not performed.
M1PE	Bus Master ID 1 Process Identifier Enable. If set, this flag specifies that the process identifier and mask defined in MPU_RGD*n*.Word3 are to be included in the region hit evaluation. If cleared, then the region hit evaluation does not include the process identifier.
M1SM	Bus Master ID 1 Supervisor Mode Access Control. This 2-bit field defines the access controls for bus master ID 1 (e200z0) when operating in supervisor mode. The M1SM field is defined as: 00 `r, w, x` = read, write and execute allowed. 01 `r, –, x` = read and execute allowed, but no write. 10 `r, w, –` = read and write allowed, but no execute. 11 Same access controls as that defined by M1UM for user mode.
M1UM	Bus Master ID 1 User Mode Access Control. This 3-bit field defines the access controls for bus master ID 1 (e200z0) when operating in user mode. The M1UM field consists of three independent bits, enabling read, write, and execute permissions: `{r,w,x}`. If set, the bit allows the given access type to occur; if cleared, an attempted access of that mode may be terminated with an access error (if not allowed by any other descriptor) and the access not performed.
M0PE	Bus Master ID 0 Process Identifier Enable. If set, this flag specifies that the process identifier and mask defined in MPU_RGD*n*.Word3 are to be included in the region hit evaluation. If cleared, the region hit evaluation does not include the process identifier.
M0SM	Bus Master ID 0 Supervisor Mode Access Control. This 2-bit field defines the access controls for bus master ID 0 (e200z6) when operating in supervisor mode. The M0SM field is defined as: 00 `r, w, x` = read, write and execute allowed. 01 `r, –, x` = read and execute allowed, but no write. 10 `r, w, –` = read and write allowed, but no execute. 11 Same access controls as that defined by M0UM for user mode.

Table 18-8. MPU_RGD*n*.Word2 Field Descriptions (continued)

Field	Description
M0UM	Bus Master ID 0 User Mode Access Control. This 3-bit field defines the access controls for bus master ID 0 (e200z6) when operating in user mode. The M0UM field consists of three independent bits, enabling read, write, and execute permissions: {r,w,x}. If set, the bit allows the given access type to occur; if cleared, an attempted access of that mode may be terminated with an access error (if not allowed by any other descriptor) and the access not performed.

18.3.2.4.4 MPU Region Descriptor *n*, Word 3 (MPU_RGD*n*.Word3)

The fourth word of the MPU region descriptor contains the optional process identifier and mask, plus the region descriptor's valid bit.

Because the region descriptor is a 128-bit entity, there are potential coherency issues as this structure is being updated because multiple writes are required to update the entire descriptor. Accordingly, the MPU hardware assists in the operation of the descriptor valid bit to prevent incoherent region descriptors from generating spurious access errors. In particular, it is expected that a complete update of a region descriptor is typically done with sequential writes to MPU_RGD*n*.Word0, then MPU_RGD*n*.Word1, ... and MPU_RGD*n*.Word3. The MPU hardware automatically clears the valid bit on any writes to words {0,1,2} of the descriptor. Writes to this word set/clear the valid bit in a normal manner.

Because it is also expected that system software may adjust the access controls within a region descriptor (MPU_RGDn.Word2) only as different tasks execute, an alternate programming view of this 32-bit entity is provided. If only the access controls are being updated, this operation must be performed by writing to MPU_RGDAAC*n* (alternate access control *n*) as stores to these locations do not affect the descriptor's valid bit.

Offset: MPU_BASE + 0x400 + (16*n) + 0xc (MPU_RGD*n*.Word3) Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	PID								PIDMASK							
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	VLD
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 18-9. MPU Region Descriptor, Word 3 Register (MPU_RGD*n*.Word3)

Table 18-9. MPU_RGD*n*.Word3 Field Descriptions

Field	Description
PID	Process Identifier. This 8-bit field specifies that the optional process identifier is to be included in the determination of whether the current access hits in the region descriptor. This field is combined with the PIDMASK and included in the region hit determination if MPU_RGD*n*.Word2[MxPE] is set. **Note:** Master ID 0 is only able to drive the process identifier of 0.
PIDMASK	Process Identifier Mask. This 8-bit field provides a masking capability so that multiple process identifiers can be included as part of the region hit determination. If a bit in the PIDMASK is set, the corresponding bit of the PID is ignored in the comparison. This field is combined with the PID and included in the region hit determination if MPU_RGD*n*.Word2[MxPE] is set. For more information on the handling of the PID and PIDMASK, see Section 18.4.1.1, Access Evaluation—Hit Determination.
VLD	Valid. This bit signals the region descriptor is valid. Any write to MPU_RGD*n*.Word{0,1,2} clears this bit, but a write to MPU_RGD*n*.Word3 sets or clears this bit depending on bit 31 of the write operand. 0 Region descriptor is invalid. 1 Region descriptor is valid.

18.3.2.5 MPU Region Descriptor Alternate Access Control *n* (MPU_RGDAAC*n*)

As noted in Section 18.3.2.4.3, MPU Region Descriptor n, Word 2 (MPU_RGDn.Word2), it is expected that because system software may adjust the access controls within a region descriptor (MPU_RGD*n*.Word2) only as different tasks execute, an alternate programming view of this 32-bit entity is desired. If only the access controls are being updated, this operation should be performed by writing to MPU_RGDAAC*n* (alternate access control *n*) as stores to these locations do not affect the descriptor's valid bit.

The memory address therefore provides an alternate location for updating MPU_RGD*n*.Word2.

Offset: MPU_BASE + 0x800 + (4**n*) (MPU_RGDAAC*n*) Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	M6RE	M6WE	M5RE	M5WE	M4RE	M4WE	0	0	0	0	0	0	M2PE	M2SM
W																
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	M2SM	M2UM			M1PE	M1SM		M1UM			M0PE	M0SM		M0UM		
W		r	w	x				r	w	x				r	w	x
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Figure 18-10. MPU RGD Alternate Access Control *n* (MPU_RGDAAC*n*)

Because the MPU_RGDAAC*n* register is another memory mapping for MPU_RGD*n*.Word2, the field definitions shown in Table 18-10 are identical to those presented in Table 18-8.

Table 18-10. MPU_RGDAACn Field Descriptions

Field	Description
M6RE	Bus Master ID 6 Read Enable. If set, this flag allows bus master ID (FlexRay) 6 to perform read operations. If cleared, any attempted read by bus master ID 6 terminates with an access error and the read is not performed.
M6WE	Bus Master ID 6 Write Enable. If set, this flag allows bus master ID 6 (FlexRay) to perform write operations. If cleared, any attempted write by bus master ID 6 terminates with an access error and the write is not performed.
M5RE	Bus Master ID 5 Read Enable. If set, this flag allows bus master ID 5 (Media Local Bus) to perform read operations. If cleared, any attempted read by bus master ID 5 terminates with an access error and the read is not performed.
M5WE	Bus Master ID 5 Write Enable. If set, this flag allows bus master ID 5 (Media Local Bus) to perform write operations. If cleared, any attempted write by bus master ID 5 terminates with an access error and the write is not performed.
M4RE	Bus Master ID 4 Read Enable. If set, this flag allows bus master ID (FEC) 4 to perform read operations. If cleared, any attempted read by bus master ID 4 terminates with an access error and the read is not performed.
M4WE	Bus Master ID 4 Write Enable. If set, this flag allows bus master ID 4 (FEC) to perform write operations. If cleared, any attempted write by bus master ID 4 terminates with an access error and the write is not performed.
M2PE	This bit can be read and written to either a 0 or 1, but the MPU behaves as if this bit was permanently tied to 0, so that the PID is not part of the region hit evaluation.
M2SM	Bus Master 2 Supervisor Mode Access Control. This 2-bit field defines the access controls for bus master ID 2 (eDMA) when operating in supervisor mode. The M2SM field is defined as: 00 `r, w, x` = read, write and execute allowed. 01 `r, –, x` = read and execute allowed, but no write. 10 `r, w, –` = read and write allowed, but no execute. 11 Same access controls as that defined by M2UM for user mode.
M2UM	Bus Master 2 User Mode Access Control. This 3-bit field defines the access controls for bus master ID 2 (eDMA) when operating in user mode. The M2UM field consists of three independent bits, enabling read, write, and execute permissions: `{r,w,x}`. If set, the bit allows the given access type to occur; if cleared, an attempted access of that mode may be terminated with an access error (if not allowed by any other descriptor) and the access not performed.
M1PE	Bus Master 1 Process Identifier Enable. If set, this flag specifies that the process identifier and mask (defined in MPU_RGD*n*.Word3) are to be included in the region hit evaluation. If cleared, the region hit evaluation does not include the process identifier.
M1SM	Bus Master 1 Supervisor Mode Access Control. This 2-bit field defines the access controls for bus master ID 1 (e200z0) when operating in supervisor mode. The M1SM field is defined as: 00 `r, w, x` = read, write and execute allowed. 01 `r, –, x` = read and execute allowed, but no write. 10 `r, w, –` = read and write allowed, but no execute. 11 Same access controls as that defined by M1UM for user mode.
M1UM	Bus Master 1 User Mode Access Control. This 3-bit field defines the access controls for bus master ID 1 (e200z0) when operating in user mode. The M1UM field consists of three independent bits, enabling read, write, and execute permissions: `{r,w,x}`. If set, the bit allows the given access type to occur; if cleared, an attempted access of that mode may be terminated with an access error (if not allowed by any other descriptor) and the access not performed.
M0PE	Bus Master 0 Process Identifier Enable. If set, this flag specifies that the process identifier and mask (defined in MPU_RGDn.Word3) are to be included in the region hit evaluation. If cleared, then the region hit evaluation does not include the process identifier.

Table 18-10. MPU_RGDAACn Field Descriptions (continued)

Field	Description
M0SM	Bus Master 0 Supervisor Mode Access Control. This 2-bit field defines the access controls for bus master ID 0 (e200z6) when operating in supervisor mode. The M0SM field is defined as: 00 `r, w, x` = read, write and execute allowed. 01 `r, –, x` = read and execute allowed, but no write. 10 `r, w, –` = read and write allowed, but no execute. 11 Same access controls as that defined by M0UM for user mode.
M0UM	Bus Master 0 User Mode Access Control. This 3-bit field defines the access controls for bus master ID 0 (e200z6) when operating in user mode. The M0UM field consists of three independent bits, enabling read, write, and execute permissions: `{r,w,x}`. If set, the bit allows the given access type to occur; if cleared, an attempted access of that mode may be terminated with an access error (if not allowed by any other descriptor) and the access not performed.

18.4 Functional Description

In this section, the functional operation of the MPU is detailed. In particular, subsequent sections discuss the operation of the access evaluation macro as well as the handling of error-terminated AHB bus cycles.

18.4.1 Access Evaluation Macro

As discussed, the basic operation of the MPU is performed in the access evaluation macro, a hardware structure replicated in the two-dimensional connection matrix. As shown in Figure 18-11, the access evaluation macro inputs the AHB system bus address phase signals (AHB_ap) and the contents of a region descriptor (RGD*n*) and performs two major functions: region hit determination (`hit_b`) and detection of an access protection violation (`error`).

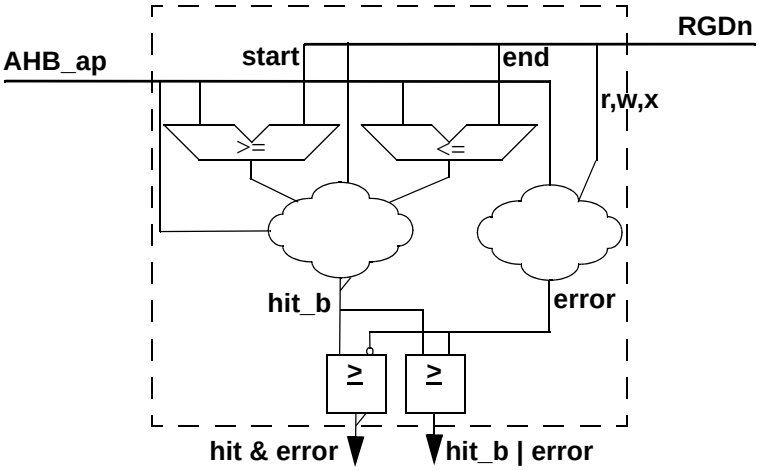


Figure 18-11. MPU Access Evaluation Macro

Figure 18-11 is not a schematic of the actual access evaluation macro, but a generalized block diagram showing the major functions included in this logic block.

18.4.1.1 Access Evaluation—Hit Determination

To determine if the current AHB reference hits in the given region, two magnitude comparators are used with the region's start and end addresses. The boolean equation for this portion of the hit determination is defined as:

```
region_hit =
        ((haddr[0:26] >= rgdn.srtaddr[0:26]) & (haddr[0:26] <= rgdn.endaddr[0:26]))
        & rgdn.vld
```

where `haddr[*]` is the current AHB reference address, `rgdn.srtaddr[*]` and `rgdn.endaddr[*]` are the start and end addresses, and `rgdn.vld` is the valid bit, all from region descriptor *n*. There are no hardware checks to verify that `rgdn.endaddr ≥ rgdn.srtaddr`, and the software must properly load appropriate values into these fields of the region descriptor.

In addition to the algebraic comparison of the AHB reference address versus the region descriptor's start and end addresses, the optional process identifier is examined against the region descriptor's PID and PIDMASK fields. Using the `hmaster[*]` number to select the appropriate M*x*PE field from the region descriptor, a process identifier hit term is formed as:

```
pid_hit = ~rgdn.mxpe
        | ((current_pid[0:7] | rgdn.pidmask[0:7]) == (rgdn.pid[0:7] | rgdn.pidmask[0:7]))
```

where the `current_pid[*]` is the selected process identifier from the current bus master, and `rgdn.pid[*]` and `rgdn.pidmask[*]` are the appropriate process identifier fields from the region descriptor *n*. For AHB bus masters that do not output a process identifier, the MPU forces the `pid_hit` term to be asserted.

As shown in Figure 18-11, the access evaluation macro forms the logical complement `(hit_b)` of the combined region_hit and pid_hit boolean equations.

18.4.1.2 Access Evaluation—Privilege Violation Determination

While the access evaluation macro is making the region hit determination, the logic is also evaluating if the current access is allowed by the permissions defined in the region descriptor. Using the AHB `hmaster[*]` and `hprot[1]` (supervisor/user mode) signals, a set of effective permissions (`eff_rgd[r,w,x]`) is generated from the appropriate fields in the region descriptor. The protection violation logic then evaluates the access against the effective permissions using the specification shown in Table 18-11.

Table 18-11. Protection Violation Definition

Description	Inputs					Output
	hwrite	*hprot[0]*	*eff_rgd[r]*	*eff_rgd[w]*	*eff_rgd[x]*	**Protection Violation?**
inst fetch read	0	0	—	—	0	yes, no x permission
inst fetch read	0	0	—	—	1	no, access is allowed
data read	0	1	0	—	—	yes, no r permission
data read	0	1	1	—	—	no, access is allowed
data write	1	—	—	0	—	yes, no w permission
data write	1	—	—	1	—	no, access is allowed

The resulting boolean equation for the processor protection violations is:

```
cpu_protection_violation
    = ~hwrite & ~hprot[0] & ~eff_rgdn[x]     // ifetch & no x
    | ~hwrite &  hprot[0] & ~eff_rgdn[r]     // data_read & no r
    |  hwrite & ~eff_rgdn[w]                 // data_write & no w
```

The resulting boolean equation for the non-processor protection violations is:

```
protection_violation
    = ~hwrite & ~eff_rgdn[r]        // data_read & no r
    |  hwrite & ~eff_rgdn[w]        // data_write & no w
```

As shown in Figure 18-11, the output of the protection violation logic is the `error` signal, that is, `error = protection_violation`.

The access evaluation macro then uses the `hit_b` and `error` signals to form two outputs. The combined (`hit_b | error`) signal is used to signal the current access is not allowed and (`~hit_b & error`) is used as the input to MPU_EDR*n* (error detail register) in the event of an error.

18.4.2 Putting It All Together and AHB Error Terminations

For each AHB MPU port being monitored, the MPU performs a reduction-AND of all the individual (`hit_b | error`) terms from each access evaluation macro. This expression then terminates the bus cycle with an error and reports a protection error for three conditions:

1. If the access does not hit in any region descriptor, a protection error is reported.
2. If the access hits in a single region descriptor and that region signals a protection violation, a protection error is reported.
3. If the access hits in multiple (overlapping) regions and all regions signal protection violations, then a protection error is reported.

The third condition reflects that priority is given to permission granting over access denying for overlapping regions as this approach provides more flexibility to system software in region descriptor assignments. For an example of the use of overlapping region descriptors, see Section 18.6, Application Information.

When the MPU causes a termination error to occur, the effect on the system depends on the bus master requesting the access. If the error was caused by a core access, a machine check is taken. If the error was caused by an eDMA access, an eDMA source or destination error occurs in the eDMA controller, which can be enabled to provide an interrupt request through the INTC. If the error was caused by a FlexRay access, a controller host interface (CHI) illegal system memory access error occurs in the FlexRay controller, which can be enabled to provide an interrupt request to the INTC.

18.5 Initialization Information

The reset state of MPU_CESR[VLD] disables the entire module. While the MPU is disabled, all accesses from all bus masters are allowed. This state also minimizes the power dissipation of the MPU. The power dissipation of each access evaluation macro is minimized when the associated region descriptor is marked as invalid or when MPU_CESR[VLD] = 0.

Typically the appropriate number of region descriptors (MPU_RGD*n*) are loaded at system startup, including the setting of the MPU_RGD*n*.Word3[VLD] bits, before MPU_CESR[VLD] is set, enabling the module. This approach allows all the loaded region descriptors to be enabled simultaneously. If a memory reference does not hit in any region descriptor, the attempted access is terminated with an error.

18.6 Application Information

In an application's system, interfacing with the MPU can generally be classified into the following activities:

1. Creation of a new memory region requires loading the appropriate region descriptor into an available register location. When a new descriptor is loaded into a RGD*n*, it would typically be performed using four 32-bit word writes. As discussed in Section 18.3.2.4.4, MPU Region Descriptor n, Word 3 (MPU_RGDn.Word3), the hardware assists in the maintenance of the valid bit, so if this approach is followed, there are no coherency issues associated with the multi-cycle descriptor writes. Deletion/removal of an existing memory region is performed by clearing MPU_RGD*n*.Word3[VLD].
2. If only the access rights for an existing region descriptor need to change, a 32-bit write to the alternate version of the access control word (MPU_RGDAAC*n*) would typically be performed. Writes to the region descriptor using this alternate access control location do not affect the valid bit, so there are, by definition, no coherency issues involved with the update. The access rights associated with the memory region switch instantaneously to the new value as the IPS write completes.
3. If the region's start and end addresses are to be changed, this would typically be performed by writing a minimum of three words of the region descriptor: MPU_RGD*n*.Word{0,1,3}, where the writes to Word0 and Word1 redefine the start and end addresses respectively and the write to Word3 re-enables the region descriptor valid bit. In many situations, all four words of the region descriptor would be rewritten.
4. Typically, references to the MPU's programming model would be restricted to supervisor mode accesses from a specific processor(s), so a region descriptor would be specifically allocated for this purpose with attempted accesses from other masters or while in user mode terminated with an error.
5. When the MPU detects an access error, the current AHB bus cycle is terminated with an error response and information on the faulting reference captured in the MPU_EAR*n* and MPU_EDR*n* registers. The error-terminated AHB bus cycle typically initiates some type of error response in the originating bus master. For example, a processor core may respond with a bus error exception, while a data movement bus master may respond with an error interrupt. In any event, the processor can retrieve the captured error address and detail information simply be reading the MPU_E{A,D}R*n* registers. Information on which error registers contain captured fault data is signaled by MPU_CESR[MPERR].
6. The process identifier seen by the MPU for master ID 0 (z6) is fixed at a value of 0. Regardless of the actual value loaded into the z6 PID0 register, the MPU always uses a value of 0 when making the optional process identifier region hit determination. This must be taken into account when configuring the associated MPU_RGDn.Word3[PID] and MPU.RGDn.Word3[PIDMASK] descriptor fields.

7. Finally, consider the use of overlapping region descriptors. Application of overlapping regions can reduce the number of descriptors required for a given set of access controls. In the overlapping memory space, the protection rights of the corresponding region descriptors are logically summed together (the boolean OR operator). In the following example of a dual-core system, there are four bus masters: the two processors (CP0, CP1) and two DMA engines (eDMA, a traditional data movement engine transferring data between RAM and peripherals, and FlexRay, a second engine transferring data to/from the RAM only). Consider the following region descriptor assignments:

Region Description	RGDn	CP0	CP1	eDMA	FlexRay	
CP0 Code	0	rwx	r--	--	--	Flash
CP1 Code	1	r--	rwx	--	--	
CP0 Data & Stack	2	rw-	---	--	--	RAM
CP0 → CP1 Shared Data	3	r--	r--	--	--	
CP1 → CP0 Shared Data						
CP0 Data & Stack	4	---	rw-	--	--	
Shared DMA Data	5	rw-	rw-	rw	rw	
MPU	6	rw-	rw-	--	--	IPS
Peripherals	7	rw-	rw-	rw	--	

Figure 18-12. Overlapping Region Descriptor Example

In this example, there are eight descriptors used to span nine regions in the three main spaces of the system memory map (flash, RAM, and IPS peripheral space). Each region indicates the specific permissions for each of the four bus masters and this definition provides an appropriate set of shared, private and executable memory spaces.

Of particular interest are the two overlapping spaces: region descriptors 2 and 3, and 3 and 4.

The space defined by RGD2 with no overlap is a private data and stack area that provides read/write access to CP0 only. The overlapping space between RGD2 and RGD3 defines a shared data space for passing data from CP0 to CP1 and the access controls are defined by the logical OR of the two region descriptors. Thus, CP0 has (`rw- | r--`) = (`rw-`) permissions, while CP1 has (`--- | r--`) = (`r--`) permission in this space. Both DMA engines are excluded from this shared processor data region. The overlapping spaces between RGD3 and RGD4 defines another shared data space, this one for passing data from CP1 to CP0. For this overlapping space, CP0 has (`r-- | ---`) = (`r--`) permission, while CP1 has (`rw- | r--`) = (`rw-`) permission. The non-overlapped space of RGD4 defines a private data and stack area for CP1 only.

The space defined by RGD5 is a shared data region, accessible by all four bus masters. Finally, the slave peripheral space mapped onto the peripheral bus is partitioned into two regions: one (RGD6) containing the MPU's programming model accessible only to the two processor cores, and the remaining peripheral region (RGD7) accessible to both processors and the traditional eDMA master.

This example is intended to show one possible application of the capabilities of the memory protection unit in a typical system.

Chapter 19
Error Correction Status Module (ECSM)

19.1 Introduction

The error correction status module (ECSM) provides a set of registers that configure and report ECC errors for the device including accesses to RAM and flash memory. The application may configure the device for the types of memory errors to be reported, and then query a set of read-only status and information registers to identify any errors that have been signaled.

There are two types of ECC errors: correctable and non-correctable. A correctable ECC error is generated when only one bit is wrong in a 64-bit doubleword. In this case, it is corrected automatically by hardware and no flags or other indication is set that the error occurred. A non-correctable ECC error is generated when two or more bits in a 64-bit doubleword are incorrect. Non-correctable ECC errors cause an interrupt, and if enabled, additional error details are available in the ECSM.

Error correction is implemented on 64 bits of data at a time, using eight bits for ECC for every 64-bit doubleword. ECC is checked on reads and calculated on writes per the following:

1. 64 bits containing the desired byte / halfword / word or doubleword in memory is read and ECC checked.
2. If the access is a write, then
   - The new byte / halfword / word / doubleword is merged into the 64 bits.
   - New ECC bits are calculated.
   - The 64 bits and the new ECC bits are written back.

NOTE

To use ECC with SRAM, the SRAM memory must be written to before ECC is enabled.

19.1.1 Features

The ECSM has this major feature:

- Registers for capturing information on platform memory errors if error-correcting codes (ECC) are implemented.

19.2 Memory Map and Registers

This section provides a detailed description of all ECSM registers.

19.2.1 Module Memory Map

The ECSM memory map is shown in Table 19-1 (a graphical layout of the registers is shown in Table 19-2 to better see Reserved areas in the memory map). The address of each register is given as an offset to the ECSM base address. Registers are listed in address order, identified by complete name and mnemonic, and lists the type of accesses allowed.

Table 19-1. ECSM Memory Map

Offset from ECSM_BASE_ADDR (0xFFF4_0000)	Register	Access	Reset Value[1]	Section/Page	Size
0x0000–0x0023	Reserved				
0x0024	FBOMCR—FEC burst optimization master control register	R/W	0x0000_0000	19.2.2.1/19-3	32
0x0028–0x0042	Reserved				
0x0043	ECR—ECC configuration register	R/W	0x00	19.2.2.2/19-5	8
0x0047	ESR—ECC status register	R/W	0x00	19.2.2.3/19-6	8
0x004A	EEGR—ECC error generation register	R/W	0x0000	19.2.2.4/19-7	16
0x0050	PFEAR—PFlash ECC address register	RO	U	19.2.2.5/19-9	32
0x0056	PFEMR—PFlash ECC master register	RO	0x0U	19.2.2.6/19-10	8
0x0057	PFEAT—PFlash ECC attributes register	RO	U	19.2.2.7/19-10	8
0x0058	PFEDRH—PFlash ECC data register high	RO	U	19.2.2.8/19-11	32
0x005C	PFEDRL—PFlash ECC data register low	RO	U	19.2.2.8/19-11	32
0x0060	PREAR—PRAM ECC address register	RO	U	19.2.2.9/19-12	32
0x0065	PRESR—PRAM ECC syndrome register	RO	U	19.2.2.10/19-13	8
0x0066	PREMR—PRAM ECC master register	RO	0x0U	19.2.2.11/19-14	8
0x0067	PREAT—PRAM ECC attributes register	RO	U	19.2.2.12/19-15	8
0x0068	PREDRH—PRAM ECC data register high	RO	U	19.2.2.13/19-16	32
0x006C	PREDRL—PRAM ECC data register low	RO	U	19.2.2.13/19-16	32
0x0007–0x3FFF	Reserved				

[1] Please refer to the register definition. U = undefined at reset.

Table 19-2. ECSM Graphical Memory Map

ECSM Offset	Register	
0x0000–0x0023	Reserved	
0x0024	FEC burst optimization master control register (FBOMCR)	
0x0028–0x003F	Reserved	
0x0040	Reserved	ECC configuration (ECR)
0x0044	Reserved	ECC status register (ESR)

Table 19-2. ECSM Graphical Memory Map

ECSM Offset	Register		
0x000048	Reserved	ECC error generation register (EEGR)	
0x004C	Reserved		
0x0050	PFlash ECC address register (PFEAR)		
0x0054	Reserved	PFlash ECC master register (PFEMR)	PFlash ECC attributes register (PFEAT)
0x0058	Reserved		
0x005C	PFlash ECC Data register (PFEDR)		
0x0060	PRAM ECC address register (PREAR)		
0x0064	Reserved	PRAM ECC master register (PREMR)	PRAM ECC attributes register (PREAT)
0x0068	Reserved		
0x006C	PRAM ECC data register (PREDR)		

19.2.2 Register Descriptions

This section lists the ECSM registers in address order and describes the registers and their bit fields. Attempted accesses to reserved addresses result in an error termination; however, attempted writes to read-only registers are ignored and do not terminate with an error.

NOTE

Unless noted otherwise, writes to the programming model must match the size of the register, e.g., an *n*-bit register only supports *n*-bit writes, etc. Attempted writes of a different size than the register width produce an error termination of the bus cycle and no change to the targeted register.

19.2.2.1 FEC Burst Optimization Master Control Register (FBOMCR)

The FEC burst optimization master control register (FBOMCR) controls FEC burst optimization behavior on the system bus. Other FEC registers are described in Section 25.3.4.3, Ethernet Interrupt Mask Register (EIMR), through Section 25.3.4.24, Receive Buffer Size Register (EMRBR).

In order to increase throughput, the FEC interface to the system bus can accumulate read requests or writes to burst those transfers on the system bus. The FBOMCR determines the XBAR ports for which this bursting is enabled, as well as whether the bursting is for reads, writes, or both. FBOMCR also controls how errors for writes are handled.

Offset: ECSM_BASE_ADDR + 0x0024 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	FXSBE0	FXSBE1	FXSBE2	FXSBE3	0	0	FXSBE6	FXSBE7	RBEN	WBEN	ACCERR	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 19-1. FEC Burst Optimization Master Control Register (FBOMCR)

Table 19-3. FBOMCR Field Descriptions

Field	Description
FXSBE*n* [0:7]	FEC XBAR slave burst enable. FXSBE*n* enables bursting by the FEC interface to the XBAR slave port controlled by that respective FXSBE*n* bit. If FXSBE*n* is asserted, then that XBAR slave port enabled by the bit can accept the bursts allowed by RBEN and WBEN. Otherwise, the FEC interface will not burst to the XBAR slave port controlled by that respective FXSBE*n* bit. Read bursts from that XBAR slave port are enabled by RBEN. Write bursts to that XBAR slave port are enabled by WBEN. FXSBE0 = Burst enable for haddr[31:29] = 3'h0 FXSBE1 = Burst enable for haddr[31:29] = 3'h1 FXSBE2 = Burst enable for haddr[31:29] = 3'h2 FXSBE3 = Burst enable for haddr[31:29] = 3'h3 FXSBE4 = Burst enable for haddr[31:29] = 3'h4 FXSBE5 = Burst enable for haddr[31:29] = 3'h5 FXSBE6 = Burst enable for haddr[31:29] = 3'h6 FXSBE7 = Burst enable for haddr[31:29] = 3'h7
RBEN	Global read burst enable from XBAR slave port designated by FXSBE*n* 0 Read bursting from all XBAR slave ports is disabled. 1 Read bursting is enabled from any XBAR slave port whose FXSBE*n* bit is asserted.
WBEN	Global write burst enable to XBAR slave port designated by FXSBE*n* 0 Write bursting to all XBAR slave ports is disabled. 1 Write bursting is enabled to any XBAR slave port whose FXSBE*n* bit is asserted.
ACCERR	Accumulate error - This bit determines whether an error response for the first half of the write burst is accumulated to the second half of the write burst or discarded. In order to complete the burst, the FEC interface to the system bus responds by indicating that the first half of the burst completed without error before it actually writes the data so that it can fetch the second half of the write data from the FIFO. When actually written onto the system bus, the first half of the write burst can have an error. Because this half initially responded without an error to the FIFO, the error is discarded or accumulated with the error response for the second half of the burst. 0 Any error to the first half of the write burst is discarded. 1 Any actual error response to the first half of the write burst is accumulated in the second half's response. In other words, an error response to the first half will be seen in the response to the second half, even if the second half does not error.

19.2.2.2 ECC Configuration Register (ECR)

The ECC configuration register is an 8-bit control register for specifying which types of memory errors are reported. In all systems with ECC, the occurrence of a non-correctable error causes the current access to be terminated with an error condition. In many cases, this error termination is reported directly by the initiating bus master. However, there are certain situations where the occurrence of this type of non-correctable error is not reported by the master. Examples include speculative instruction fetches that are discarded due to a change-of-flow operation and buffered operand writes. The ECC reporting logic in the ECSM provides an optional error interrupt mechanism to signal all non-correctable memory errors. In addition to the interrupt generation, the ECSM captures specific information (memory address, attributes and data, bus master number, etc.) that may be useful for subsequent failure analysis.

See Figure 19-2 and Table 19-4 for the ECC configuration register definition.

Offset: ECSM_BASE_ADDR + 0x0043 Access: User read/write

	0	1	2	3	4	5	6	7
R	0	0	EPR1BR	EPF1BR	0	0	EPRNCR	EPFNCR
W								
Reset	0	0	0	0	0	0	0	0

Figure 19-2. ECC Configuration (ECR) Register

Table 19-4. ECR Field Descriptions

Field	Description
EPR1BR	Enable Platform RAM 1-bit Reporting. The occurrence of a single-bit RAM correction generates an ECSM ECC interrupt request as signalled by the assertion of ESR[PR1BC]. The address, attributes and data are also captured in the PREAR, PRESR, PREMR, PREAT, and PREDR registers. 0 Reporting of single-bit platform RAM corrections is disabled. 1 Reporting of single-bit platform RAM corrections is enabled.
EPF1BR	Enable Platform Flash 1-bit Reporting. The occurrence of a single-bit flash correction generates an ECSM ECC interrupt request as signalled by the assertion of ESR[PF1BC]. The address, attributes, and data are also captured in the PFEAR, PFEMR, PFEAT, and PFEDR registers. 0 Reporting of single-bit platform flash corrections is disabled. 1 Reporting of single-bit platform flash corrections is enabled.
EPRNCR	Enable Platform RAM Non-Correctable Reporting. The occurrence of a non-correctable multi-bit RAM error generates an ECSM ECC interrupt request as signaled by the assertion of ESR[PRNCE]. The faulting address, attributes, and data in either the 512 KB or 80 KB array are also captured in the PREAR, PRESR, PREMR, PREAT, and PREDR registers. 0 Reporting of non-correctable platform RAM errors is disabled. 1 Reporting of non-correctable platform RAM errors is enabled.
EPFNCR	Enable Platform Flash Non-Correctable Reporting. The occurrence of a non-correctable multi-bit flash error generates an ECSM ECC interrupt request as signaled by the assertion of ESR[PFNCE]. The faulting address, attributes, and data are also captured in the PFEAR, PFEMR, PFEAT, and PFEDR registers. 0 Reporting of non-correctable platform flash errors is disabled. 1 Reporting of non-correctable platform flash errors is enabled.

19.2.2.3 ECC Status Register (ESR)

The ECC status register is an 8-bit control register for signaling which types of properly enabled ECC events have been detected. The ESR signals the last properly enabled memory event to be detected. An ECC interrupt request is asserted if any flag bit is asserted and its corresponding enable bit is asserted.

The ECSM allows a maximum of one bit of the ESR to be asserted at any given time. This preserves the association between the ESR and the corresponding address and attribute registers, which are loaded on each occurrence of an properly enabled ECC event. If there is a pending ECC interrupt and another properly enabled ECC event occurs, the ECSM hardware automatically handles the ESR reporting, clearing the previous data and loading the new state and thus guaranteeing that only a single flag is asserted.

To maintain the coherent software view of the reported event, the following sequence in the ECSM error interrupt service routine is suggested:

1. Read the ESR and save it.
2. Read and save all the address and attribute reporting registers.
3. Re-read the ESR and verify the current contents matches the original contents. If the two values are different, repeat from step one.
4. When the values are identical, write a 1 to the asserted ESR flag to negate the interrupt request.

See Figure 19-3 and Table 19-5 for the ECC status register definition.

Offset: ECSM_BASE_ADDR + 0x0047 Access: User read/write

	0	1	2	3	4	5	6	7
R	0	0	PR1BC	PF1BC	0	0	PRNCE	PFNCE
W			w1c	w1c			w1c	w1c
Reset	0	0	0	0	0	0	0	0

Figure 19-3. ECC Status (ESR) Register

Table 19-5. ESR Field Descriptions

Field	Description
PR1BC	Platform RAM 1-bit Correction. This bit can only be set when ECR[EPR1BR] is asserted. The occurrence of a properly-enabled single-bit RAM correction generates an ECSM ECC interrupt request. The address, attributes and data are also captured in the PREAR, PRESR, PREMR, PREAT, and PREDR registers. To clear this interrupt flag, write a 1 to this bit. Writing a 0 has no effect. 0 No reportable single-bit platform RAM correction has been detected. 1 A reportable single-bit platform RAM correction has been detected.
PF1BC	Platform Flash 1-bit Correction. This bit can only be set when ECR[EPF1BR] is asserted. The occurrence of a properly-enabled single-bit flash correction generates an ECSM ECC interrupt request. The address, attributes and data are also captured in the PFEAR, PFEMR, PFEAT, and PFEDR registers. To clear this interrupt flag, write a 1 to this bit. Writing a 0 has no effect. 0 No reportable single-bit platform flash correction has been detected. 1 A reportable single-bit platform flash correction has been detected.

Table 19-5. ESR Field Descriptions (continued)

Field	Description
PRNCE	Platform RAM Non-Correctable Error. The occurrence of a properly enabled non-correctable RAM error generates an ECSM ECC interrupt request. The faulting address, attributes, and data in either the 512K or 80K array are also captured in the PREAR, PRESR, PREMR, PREAT, and PREDR registers. To clear this interrupt flag, write a 1 to this bit. Writing a 0 has no effect. 0 No reportable non-correctable platform RAM error has been detected. 1 A reportable non-correctable platform RAM error has been detected.
PFNCE	Platform Flash Non-Correctable Error. The occurrence of a properly enabled non-correctable flash error generates an ECSM ECC interrupt request. The faulting address, attributes and data are also captured in the PFEAR, PFEMR, PFEAT, and PFEDR registers. To clear this interrupt flag, write a 1 to this bit. Writing a 0 has no effect. 0 No reportable non-correctable platform flash error has been detected. 1 A reportable non-correctable platform flash error has been detected.

If both a flash and RAM non-correctable error occur at the same time, the ECSM records the event with the PR1BC as highest priority, then PF1BC, then, PRNCE, and finally PFNCE. If both a 512 KB and an 80 KB RAM non-correctable error occur at the same time, the ECSM records the event with the 512 KB array.

19.2.2.4 ECC Error Generation Register (EEGR)

The ECC error generation register is a 16-bit control register used to force the generation of single- and double-bit data inversions in the platform memories with ECC, most notably the platform RAM. This capability is provided for two purposes:

- It provides a software-controlled mechanism for injecting errors into the platform memories during data writes to verify the integrity of the ECC logic.
- It provides a mechanism to allow testing of the software service routines associated with memory error logging.

The intent is to generate errors during data write cycles, such that subsequent reads of the corrupted address locations generate ECC events, either single-bit corrections or double-bit noncorrectable errors that are terminated with an error response.

See Figure 19-4 and Table 19-6 for the ECC error generation register definition.

Offset: ECSM_BASE_ADDR + 0x004A — Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	FRC1BI	FR11BI	0	0	FRCNCI	FR1NCI	PREI_SEL	ERRBIT						
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 19-4. ECC Error Generation (EEGR) Register

Table 19-6. EEGR Field Descriptions

Field	Description
FRC1BI	Force Platform RAM Continuous 1-Bit Data Inversions. The assertion of this bit forces the platform RAM controller to create 1-bit data inversions, as defined by the bit position specified in ERRBIT[6:0], continuously on every write operation. The normal ECC generation takes place in the PRAM controller, but then the polarity of the bit position defined by ERRBIT is inverted to introduce a 1-bit ECC event in the RAM. After this bit has been enabled to generate another continuous 1-bit data inversion, it must be cleared before being set again to properly re-enable the error generation logic. 0 No platform RAM continuous 1-bit data inversions are generated. 1 1-bit data inversions in the platform RAM are continuously generated.
FR11BI	Force Platform RAM One 1-bit Data Inversion. The assertion of this bit forces the platform RAM controller to create one 1-bit data inversion, as defined by the bit position specified in ERRBIT[6:0], on the first write operation after this bit is set. The normal ECC generation takes place in the PRAM controller, but then the polarity of the bit position defined by ERRBIT is inverted to introduce a 1-bit ECC event in the RAM. After this bit has been enabled to generate a single 1-bit data inversion, it must be cleared before being set again to properly re-enable the error generation logic. 0 No platform RAM single 1-bit data inversion is generated. 1 One 1-bit data inversion in the platform RAM is generated.
FRCNCI	Force Platform RAM Continuous Noncorrectable Data Inversions. The assertion of this bit forces the platform RAM controller to create 2-bit data inversions, as defined by the bit position specified in ERRBIT and the overall odd parity bit, continuously on every write operation. After this bit has been enabled to generate another continuous noncorrectable data inversion, it must be cleared before being set again to properly re-enable the error generation logic. The normal ECC generation takes place in the PRAM controller, but then the polarity of the bit position defined by ERRBIT and the overall odd parity bit are inverted to introduce a 2-bit ECC error in the RAM. 0 No platform RAM continuous 2-bit data inversions are generated. 1 2-bit data inversions in the platform RAM are continuously generated.
FR1NCI	Force Platform RAM One Noncorrectable Data Inversions. The assertion of this bit forces the platform RAM controller to create one 2-bit data inversion, as defined by the bit position specified in ERRBIT and the overall odd parity bit, on the first write operation after this bit is set. The normal ECC generation takes place in the PRAM controller, but then the polarity of the bit position defined by ERRBIT and the overall odd parity bit are inverted to introduce a 2-bit ECC error in the RAM. After this bit has been enabled to generate a single 2-bit error, it must be cleared before being set again to properly re-enable the error generation logic. 0 No platform RAM single 2-bit data inversions are generated. 1 One 2-bit data inversion in the platform RAM is generated.

Table 19-6. EEGR Field Descriptions (continued)

Field	Description
PREI_SEL	Platform RAM Error Injection Select. Platform RAM Error Injection Select. The platform contains two platform RAM blocks with ECC. This bit selects which RAM is injected. 0 PRAM0 is injected. 1 PRAM1 is injected.

ERRBIT	Error Bit Position. The vector defines the bit position, which is complemented to create the data inversion on the write operation. For the creation of 2-bit data inversions, the bit specified by this field plus the odd parity bit of the ECC code are inverted.
	The platform RAM controller follows a vector bit ordering scheme where LSB = 0. Errors in the ECC syndrome bits can be generated by setting this field to a value greater than the RAM width. For example, consider a 32-bit RAM implementation.
	The 32-bit ECC approach requires seven code bits for a 32-bit word. For PRAM data width of 32 bits, the actual SRAM (32b data + 7b for ECC = 39 bits. The following association between the ERRBIT field and the corrupted memory bit is defined:
	`if ERRBIT = 0, then RAM[0] is inverted`
	`if ERRBIT = 1, then RAM[1] is inverted`
	`...`
	`if ERRBIT = 31, then RAM[31] is inverted`
	`if ERRBIT = 64,then ECC Parity[0] is inverted`
	`if ERRBIT = 65,then ECC Parity[1] is inverted`
	`...`
	`if ERRBIT = 70,then ECC Parity[6] is inverted`
	Note: For ERRBIT values of 32 to 63 and greater than 70, no bit position is inverted.

NOTE

If an attempt to force a non-correctable inversion by asserting EEGR[FRCNCI] or EEGR[FRC1NCI], and EEGR[ERRBIT] equals 64, no data inversion is generated.

The only allowable values for the 4 control bit enables {FR11BI, FRC1BI, FRCNCI, FR1NCI} are {0,0,0,0}, {1,0,0,0}, {0,1,0,0}, {0,0,1,0} and {0,0,0,1}. All other values result in undefined behavior.

19.2.2.5 Platform Flash ECC Address Register (PFEAR)

The PFEAR is a 32-bit register for capturing the address of the last properly enabled ECC event in the platform flash memory. Depending on the state of the ECC configuration register, an ECC event in the platform flash causes the address, attributes and data associated with the access to be loaded into the PFEAR, PFEMR, PFEAT, and PFEDR registers and also the appropriate flag (PF1BC or PFNCE) in the ECC status register to be asserted.

This register is read-only; any attempted write is ignored. See Figure 19-5 and Table 19-7 for the platform flash ECC address register definition.

Offset: ECSM_BASE_ADDR + 0x0050 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	PFEAR															
W																
Reset	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	PFEAR															
W																
Reset	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U

Figure 19-5. Platform Flash ECC Address (PFEAR) Register

Table 19-7. PFEAR Field Descriptions

Field	Description
PFEAR	Platform Flash ECC Address Register. Contains the faulting access address of the last properly enabled platform flash ECC event.

19.2.2.6 Platform Flash ECC Master Number Register (PFEMR)

The PFEMR is a 4-bit register for capturing the AXBS bus master number of the last properly enabled ECC event in the platform flash memory. Depending on the state of the ECC configuration register, an ECC event in the platform flash causes the address, attributes and data associated with the access to be loaded into the PFEAR, PFEMR, PFEAT, and PFEDR registers and also the appropriate flag (PF1BC or PFNCE) in the ECC status register to be asserted.

This register is read-only; any attempted write is ignored. See Figure 19-6 and Table 19-8 for the platform flash ECC master number register definition.

Offset: ECSM_BASE_ADDR + 0x0056 Access: User read-only

	0	1	2	3	4	5	6	7
R	0	0	0	0	PFEMR			
W								
Reset	0	0	0	0	U	U	U	U

Figure 19-6. Platform Flash ECC Master Number (PFEMR) Register

Table 19-8. PFEMR Field Descriptions

Field	Description
PFEMR	Platform Flash CC Master Number Register. Contains the AXBS bus master number of the faulting access of the last properly enabled platform flash ECC event.

19.2.2.7 Platform Flash ECC Attributes Register (PFEAT)

The PFEAT is an 8-bit register for capturing the AXBS bus master attributes of the last properly enabled ECC event in the platform flash memory. Depending on the state of the ECC configuration register, an

ECC event in the platform flash causes the address, attributes and data associated with the access to be loaded into the PFEAR, PFEMR, PFEAT, and PFEDR registers and also the appropriate flag (PF1BC or PFNCE) in the ECC status register to be asserted.

This register is read-only; any attempted write is ignored. See Figure 19-7 and Table 19-9 for the platform flash ECC attributes register definition.

Offset: ECSM_BASE_ADDR + 0x0057 Access: User read-only

	0	1	2	3	4	5	6	7
R	WRITE	SIZE			PROTECTION			
W								
Reset	U	U	U	U	U	U	U	U

Figure 19-7. Platform Flash ECC Attributes (PFEAT) Register

Table 19-9. PFEAT Field Descriptions

Field	Description
WRITE	0 Read access. 1 Write access.
SIZE	000 8-bit access 001 16-bit access 010 32-bit access 011 64-bit access 1xx Reserved
PROTECTION	Cache: 0xxx Non-cacheable 1xxx Cacheable Buffer: x0xx Non-bufferable x1xx Bufferable Mode: xx0x User mode xx1x Supervisor mode Type: xxx0 I-Fetch xxx1 Data

19.2.2.8 Platform Flash ECC Data Register (PFEDR)

The PFEDR is a 64-bit register for capturing the data associated with the last properly enabled ECC event in the platform flash memory. Depending on the state of the ECC configuration register, an ECC event in the platform flash causes the address, attributes and data associated with the access to be loaded into the PFEAR, PFEMR, PFEAT, and PFEDR registers and also the appropriate flag (PF1BC or PFNCE) in the ECC status register to be asserted.

The data captured on a multi-bit non-correctable ECC error is undefined.

This register is read-only; any attempted write is ignored. See Figure 19-9 and Table 19-10 for the platform flash ECC data register definition.

Offset: ECSM_BASE_ADDR + 0x0058 Access: User read-only

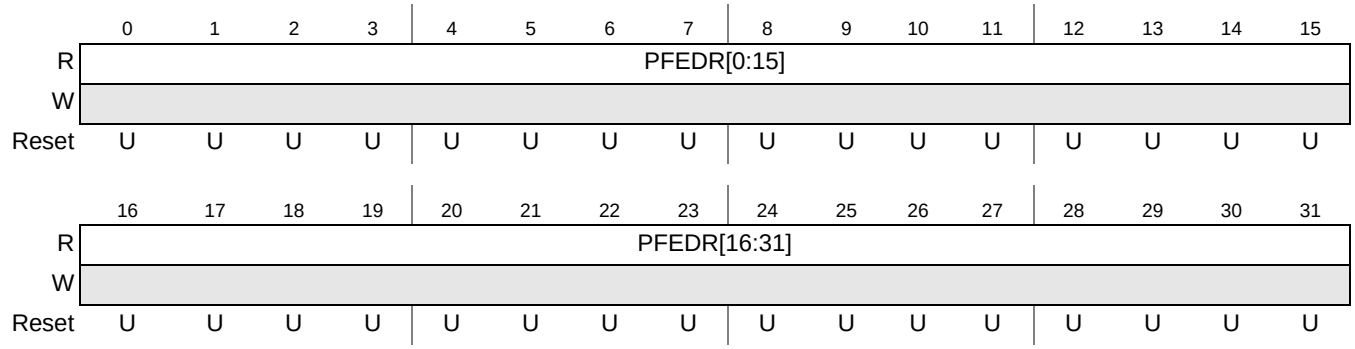


Figure 19-8. Platform Flash ECC Data High (PFEDRH) Register

Offset: ECSM_BASE_ADDR + 0x005C Access: User read-only

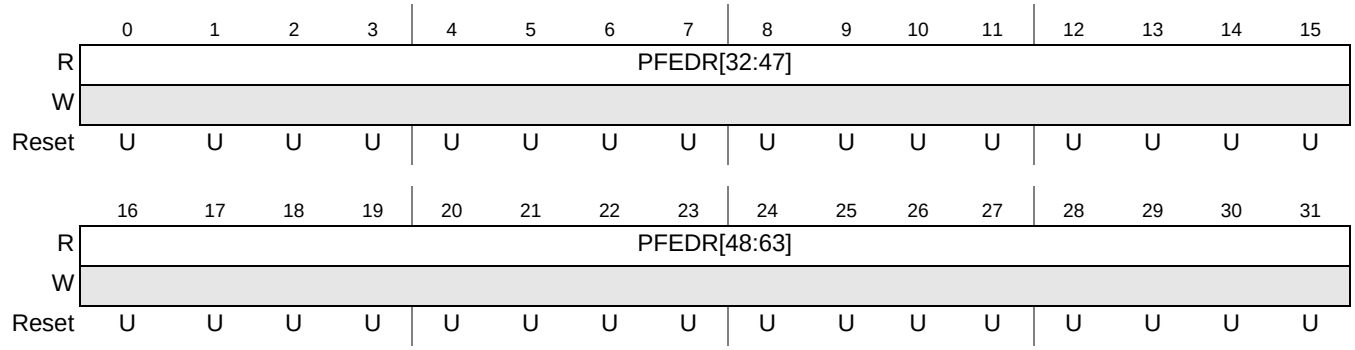


Figure 19-9. Platform Flash ECC Data Low (PFEDRL) Register

Table 19-10. PFEDR Field Descriptions

Field	Description
PFEDR	Platform Flash ECC Data Register. Contains the data associated with the faulting access of the last properly enabled platform flash ECC event. The register contains the data value taken directly from the platform data bus.

19.2.2.9 Platform RAM ECC Address Register (PREAR)

The PREAR is a 32-bit register for capturing the address of the last properly enabled ECC event in the platform RAM memory. Depending on the state of the ECC configuration register, an ECC event in the platform RAM causes the address, attributes and data associated with the access to be loaded into the PREAR, PRESR, PREMR, PREAT, and PREDR registers and also the appropriate flag (PR1BC or PRNCE) in the ECC status register to be asserted.

This register is read-only; any attempted write is ignored. See Figure 19-10 and Table 19-11 for the PREAR definition.

Offset: ECSM_BASE_ADDR + 0x0060 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	PREAR															
W																
Reset	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	PREAR															
W																
Reset	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U

Figure 19-10. Platform RAM ECC Address (PREAR) Register

Table 19-11. PREAR Field Descriptions

Field	Description
PREAR	Platform RAM ECC Address Register. Contains the faulting access address of the last properly enabled platform RAM ECC event.

19.2.2.10 Platform RAM ECC Syndrome Register (PRESR)

The PRESR is an 8-bit register for capturing the error syndrome of the last properly enabled ECC event in the platform RAM memory. Depending on the state of the ECC configuration register, an ECC event in the platform RAM causes the address, attributes, and data associated with the access to be loaded into the PREAR, PRESR, PREMR, PREAT and PREDR registers, and the appropriate flag (PR1BC or PRNCE) in the ECC status register to be asserted.

This register is read-only; any attempted write is ignored. See Figure 19-11 and Table 19-12 for the Platform RAM ECC syndrome register definition.

Offset: ECSM_BASE_ADDR + 0x0065 Access: User read-only

	0	1	2	3	4	5	6	7
R	PRESR[0:7]							
W								
Reset	U	U	U	U	U	U	U	U

Figure 19-11. Platform RAM ECC Syndrome (PRESR) Register

Table 19-12. PRESR Field Descriptions

Field	Description
PRESR	Platform RAM ECC Syndrome Register. This 8-bit syndrome field includes 7 bits of Hamming decoded parity plus an odd-parity bit for the entire 72-bit (64-bit data + 8 ECC) code word. The upper 7 bits of the syndrome specify the exact bit position in error for single-bit correctable codewords, and the combination of a non-zero 7-bit syndrome plus overall incorrect parity bit signal a multi-bit, non-correctable error. For correctable single-bit errors, the mapping shown in Table 19-13 associates the upper 7 bits of the syndrome with the data bit in error.

Table 19-13. Platform RAM Syndrome Mapping for Single-Bit Correctable Errors

PRESR[0:7]	Data Bit in Error	PRESR[0:7]	Data Bit in Error	PRESR[0:7]	Data Bit in Error
0x00	No Error	0x4F	DATA[32]	0xA4	DATA[41]
0x01	ECC[0]	0x52	DATA[34]	0xA7	DATA[42]
0x02	ECC[1]	0x54	DATA[35]	0xA8	DATA[43]
0x04	ECC[2]	0x57	DATA[36]	0xAB	DATA[44]
0x08	ECC[3]	0x58	DATA[37]	0xAD	DATA[45]
0x0B	DATA[17]	0x5B	DATA[38]	0xB0	DATA[46]
0x0E	DATA[16]	0x5D	DATA[39]	0xB5	DATA[47]
0x10	ECC[4]	0x62	DATA[56]	0xCB	DATA[1]
0x13	DATA[18]	0x64	DATA[57]	0xCE	DATA[0]
0x15	DATA[19]	0x67	DATA[58]	0xD3	DATA[2]
0x16	DATA[20]	0x68	DATA[59]	0xD5	DATA[3]
0x19	DATA[21]	0x6B	DATA[60]	0xD6	DATA[4]
0x1A	DATA[22]	0x6D	DATA[61]	0xD9	DATA[5]
0x1C	DATA[23]	0x70	DATA[62]	0xDA	DATA[6]
0x20	ECC[5]	0x75	DATA[63]	0xDC	DATA[7]
0x23	DATA[8]	0x80	ECC[7]	0xE3	DATA[24]
0x25	DATA[9]	0x8A	DATA[49]	0xE5	DATA[25]
0x26	DATA[10]	0x8F	DATA[48]	0xE6	DATA[26]
0x29	DATA[11]	0x92	DATA[50]	0xE9	DATA[27]
0x2A	DATA[12]	0x94	DATA[51]	0xEA	DATA[28]
0x2C	DATA[13]	0x97	DATA[52]	0xEC	DATA[29]
0x31	DATA[14]	0x98	DATA[53]	0xF1	DATA[30]
0x34	DATA[15]	0x9B	DATA[54]	0xF4	DATA[31]
0x40	ECC[6]	0x9D	DATA[55]	Other values	Multiple bit error
0x4A	DATA[33]	0xA2	DATA[40]		

19.2.2.11 Platform RAM ECC Master Number Register (PREMR)

The PREMR is a 4-bit register for capturing the AXBS bus master number of the last properly enabled ECC event in the platform RAM memory. Depending on the state of the ECC configuration register, an ECC event in the platform RAM causes the address, attributes, and data associated with the access to be loaded into the PREAR, PRESR, PREMR, PREAT, and PREDR registers and also the appropriate flag (PR1BC or PRNCE) in the ECC status register to be asserted.

This register is read-only; any attempted write is ignored. See Figure 19-12 and Table 19-14 for the Platform RAM ECC master number register definition.

Offset: ECSM_BASE_ADDR + 0x0066 Access: User read-only

	0	1	2	3	4	5	6	7
R	0	0	0	0	PREMR			
W								
Reset	0	0	0	0	U	U	U	U

Figure 19-12. Platform RAM ECC Master Number (PREMR) Register

Table 19-14. PREMR Field Descriptions

Field	Description
PREMR	Platform RAM ECC Master Number Register. Contains the AXBS bus master number of the faulting access of the last properly enabled platform RAM ECC event.

19.2.2.12 Platform RAM ECC Attributes Register (PREAT)

The PREAT is an 8-bit register for capturing the AXBS bus master attributes of the last properly enabled ECC event in the platform RAM memory. Depending on the state of the ECC configuration register, an ECC event in the platform RAM causes the address, attributes, and data associated with the access to be loaded into the PREAR, PRESR, PREMR, PREAT, and PREDR registers and also the appropriate flag (PR1BC or PRNCE) in the ECC status register to be asserted.

This register is read-only; any attempted write is ignored. See Figure 19-13 and Table 19-15 for the platform RAM ECC attributes register definition.

Offset: ECSM_BASE_ADDR + 0x0067 Access: User read-only

	0	1	2	3	4	5	6	7
R	WRITE	SIZE			PROTECTION			
W								
Reset	U	U	U	U	U	U	U	U

Figure 19-13. Platform RAM ECC Attributes (PREAT) Register

Table 19-15. PREAT Field Descriptions

Field	Description
WRITE	0 Read access. 1 Write access.

Table 19-15. PREAT Field Descriptions (continued)

Field	Description
SIZE	000 8-bit access. 001 16-bit access. 010 32-bit access. 011 64-bit access. 1xx Reserved.
PROTECTION	Cache: 0xxx Non-cacheable. 1xxx Cacheable. Buffer: x0xx Non-bufferable. x1xx Bufferable. Mode: xx0x User mode. xx1x Supervisor mode. Type: xxx0 I-Fetch. xxx1 Data.

19.2.2.13 Platform RAM ECC Data Register (PREDR)

The PREDR is a 64-bit register for capturing the data associated with the last properly enabled ECC event in the platform RAM memory. Depending on the state of the ECC configuration register, an ECC event in the platform RAM causes the address, attributes, and data associated with the access to be loaded into the PREAR, PRESR, PREMR, PREAT, and PREDR registers and also the appropriate flag (PR1BC or PRNCE) in the ECC status register to be asserted.

The data captured on a multi-bit non-correctable ECC error is undefined. This register is read-only; any attempted write is ignored. See Figure 19-15 and Table 19-16 for the platform RAM ECC data register definition.

Offset: ECSM_BASE_ADDR + 0x0068 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	PREDR[0:15]															
W																
Reset	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	PREDR[16:31]															
W																
Reset	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U

Figure 19-14. Platform RAM ECC Data High (PREDRH) Register

Offset: ECSM_BASE_ADDR + 0x006C Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	PREDR[32:47															
W																
Reset	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	PREDR48:63]															
W																
Reset	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U

Figure 19-15. Platform RAM ECC Data Low (PREDRL) Register

Table 19-16. PREDR Field Descriptions

Field	Description
PREDR	Platform RAM ECC Data Register. Contains the data associated with the faulting access of the last properly enabled platform RAM ECC event. The register contains the data value taken directly from the platform data bus.

Chapter 20
Software Watchdog Timer (SWT)

20.1 Introduction

The Software Watchdog Timer (SWT) is a peripheral module that can prevent system lockup in situations such as software getting trapped in a loop or if a bus transaction fails to terminate. When enabled, the SWT requires periodic execution of a watchdog servicing operation. The servicing operation resets the timer to a specified time-out period. If this servicing action does not occur before the timer expires, the SWT generates an interrupt or hardware reset. The SWT can be configured to generate a reset or interrupt on an initial time-out. A reset is always generated on a second consecutive time-out.

The SWT is clocked only from the 16 MHz IRC clock. This clock source is independent from the other system clocks and hence offers an improved level of safety, since supporting only a single clock source eliminates any risk of incorrect clock selection.

The SWT is reset in Sleep mode. The user can select whether the SWT runs in other modes with the SIU_HLT1 register.

20.1.1 Features

The SWT has the following features:

- 32-bit time-out register to set the time-out period
- Programmable selection of window mode or regular servicing
- Programmable selection of reset or interrupt on an initial time-out
- Programmable selection of fixed or keyed servicing
- Master access protection
- Hard and soft configuration lock bits

20.1.2 Modes of Operation

The SWT supports two device modes of operation: normal and debug. When the SWT is enabled in normal mode, its counter runs continuously. In debug mode, operation of the counter is controlled by the FRZ bit in the SWT_CR. If the FRZ bit is set, the counter is stopped in debug mode, otherwise it continues to run.

20.2 External Signal Description

The SWT module does not have any external interface signals.

20.3 Memory Map and Register Definition

The SWT programming model has seven 32-bit registers. The programming model can only be accessed using 32-bit (word) accesses. References using a different size are invalid. Other types of invalid accesses include: writes to read-only registers, incorrect values written to the service register when enabled, accesses to reserved addresses and accesses by masters without permission. If the RIA bit in the SWT_CR is set, then the SWT generates a system reset on an invalid access. Otherwise, a bus error is generated. If either the HLK or SLK bits in the SWT_CR are set, then the SWT_CR, SWT_TO, SWT_WN, and SWT_SK registers are read-only.

20.3.1 Memory Map

The SWT memory map is shown in Table 20-1.

Table 20-1. SWT Memory Map

Offset from SWT_BASE (0xFFF3_8000)	Register	Access	Reset Value	Section/Page
0x0000	SWT_CR – SWT control register	R/W	0xFF00_0103	20.3.2.1/20-2
0x0004	SWT_IR – SWT interrupt register	R/W	0x0000_0000	20.3.2.2/20-4
0x0008	SWT_TO – SWT time-out register	R/W	0x0002_7100	20.3.2.3/20-4
0x000C	SWT_WN – SWT window register	R/W	0x0000_0000	20.3.2.4/20-5
0x0010	SWT_SR – SWT service register	R/W	0x0000_0000	20.3.2.5/20-6
0x0014	SWT_CO – SWT counter output register	R	0x0000_0000	20.3.2.6/20-6
0x0018	SWT_SK – SWT service key register	R/W	0x0000_0000	20.3.2.7/20-7
0x001C– 0x3FFF	Reserved			

20.3.2 Register Descriptions

The following sections detail the individual registers within the SWT.

20.3.2.1 SWT Control Register (SWT_CR)

The SWT_CR contains fields for configuring and controlling the SWT. This register is read-only if either the SWT_CR[HLK] or SWT_CR[SLK] bits are set.

Offset: SWT_BASE + 0x0000 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	MAP 0	MAP 1	MAP 2	MAP 3	MAP 4	MAP 5	MAP 6	MAP 7	0	0	0	0	0	0	0	0
W																
Reset	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	KEY	RIA	WND	ITR	HLK	SLK	0	0	FRZ	WEN
W																
Reset	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	1

Figure 20-1. SWT Control Register (SWT_CR)

Table 20-2. SWT_CR Field Descriptions

Field	Description
MAP*n*	Master Access Protection for Master *n*. The PXN20 bus master assignments are shown in the following table. 0 Access for the master is not enabled 1 Access for the master is enabled. **Bit** / **Bus Master** / **Bit** / **Bus Master** MAP0 / 0 — Z6 Core / MAP4 / 4 — FEC MAP1 / 1 — Z0 Core / MAP5 / 5 — MLB MAP2 / 2 — eDMA / MAP6 / 6 — FlexRay MAP3 / 3 — not used / MAP7 / 7 — not used
KEY	Keyed Service Mode. 0 Fixed Service Sequence, the fixed sequence 0xA602, 0xB480 is used to service the watchdog. 1 Keyed Service Mode, two pseudorandom key values are used to service the watchdog.
RIA	Reset on Invalid Access. 0 Invalid access to the SWT generates a bus error. 1 Invalid access to the SWT causes a system reset if WEN = 1.
WND	Window Mode. 0 Regular mode, service sequence can be done at any time. 1 Windowed mode, the service sequence is only valid when the down counter is less than the value in the SWT_WN register.
ITR	Interrupt Then Reset. 0 Generate a reset on a time-out. 1 Generate an interrupt on an initial time-out, reset on a second consecutive time-out.
HLK	Hard Lock. This bit is only cleared at reset. 0 SWT_CR, SWT_TO, SWT_WN and SWT_SK are read/write registers if SLK = 0. 1 SWT_CR, SWT_TO, SWT_WN and SWT_SK are read-only registers.

Table 20-2. SWT_CR Field Descriptions

Field	Description
SLK	Soft Lock. This bit is cleared by writing the unlock sequence to the service register. 0 SWT_CR, SWT_TO SWT_WN and SWT_SK are read/write registers if HLK = 0. 1 SWT_CR, SWT_TO, SWT_WN and SWT_SK are read-only registers.
FRZ	Debug Mode Control. Allows the watchdog timer to be stopped when the device enters debug mode. 0 SWT counter continues to run in debug mode. 1 SWT counter is stopped in debug mode.
WEN	Watchdog Enabled. 0 SWT is disabled. 1 SWT is enabled.

20.3.2.2 SWT Interrupt Register (SWT_IR)

The SWT_IR contains the time-out interrupt flag.

Offset: SWT_BASE + 0x0004 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	TIF
W																w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 20-2. SWT Interrupt Register (SWT_IR)

Table 20-3. SWT_IR Field Descriptions

Field	Description
TIF	Time-out Interrupt Flag. The flag and interrupt are cleared by writing a 1 to this bit. Writing a 0 has no effect. 0 No interrupt request. 1 Interrupt request due to an initial time-out.

20.3.2.3 SWT Time-Out Register (SWT_TO)

The SWT Time-Out (SWT_TO) register contains the 32-bit time-out period. This register is read-only if either the SWT_CR[HLK] or SWT_CR[SLK] bits are set.

Offset: SWT_BASE + 0x0008 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	WTO															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	WTO															
Reset	0	1	1	1	0	0	0	1	0	0	0	0	0	0	0	0

Figure 20-3. SWT Time-Out Register (SWT_TO)

Table 20-4. SWT_TO Register Field Descriptions

Field	Description
WTO	Watchdog time-out period in clock cycles. An internal 32-bit down counter is loaded with this value or 0x0100, whichever is greater, when the service sequence is written or when the SWT is enabled.

20.3.2.4 SWT Window Register (SWT_WN)

The SWT Window (SWT_WN) register contains the 32-bit window start value. This register is cleared on reset. This register is read-only if either the SWT_CR[HLK] or SWT_CR[SLK] bits are set.

Offset: SWT_BASE + 0x000C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	WST															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	WST															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 20-4. SWT Window Register (SWT_WN)

Table 20-5. SWT_WN Register Field Descriptions

Field	Description
WST	Window start value. When window mode is enabled, the service sequence can only be written when the internal down counter is less than this value.

20.3.2.5 SWT Service Register (SWT_SR)

The SWT Time-Out (SWT_SR) service register is the target for service operation writes used to reset the watchdog timer.

Offset: SWT_BASE + 0x0010 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	WSC															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 20-5. SWT Service Register (SWT_SR)

Table 20-6. SWT_SR Field Descriptions

Field	Description
WSC	Watchdog Service Code.This field is used to service the watchdog and to clear the soft lock bit (SWT_CR[SLK]). If the SWT_CR[KEY] bit is set, two pseudorandom key values are written to service the watchdog, see Section Section 20.4, Functional Description, for details. Otherwise, the sequence 0xA602 followed by 0xB480 is written to the WSC field. To clear the soft lock bit (SWT_CR[SLK]), the value 0xC520 followed by 0xD928 is written to the WSC field.

20.3.2.6 SWT Counter Output Register (SWT_CO)

The SWT Counter Output (SWT_CO) register is a read-only register that shows the value of the internal down counter when the SWT is disabled.

Offset: SWT_BASE + 0x0014 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CNT															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CNT															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 20-6. SWT Counter Output Register (SWT_CO)

Table 20-7. SWT_CO Register Field Descriptions

Field	Description
CNT	Watchdog Count. When the watchdog is disabled (SWT_CR[WEN] = 0) this field shows the value of the internal down counter. When the watchdog is enabled the value of this field is 0x0000_0000. Values in this field can lag behind the internal counter value for as many as 6 system plus 8 counter clock cycles. Therefore, the value read from this field immediately after disabling the watchdog may be higher than the actual value of the internal counter.

20.3.2.7 SWT Service Key Register (SWT_SK)

The SWT Service Key (SWT_SK) register holds the previous (or initial) service key value. This register is read-only if either the SWT_CR[HLK] or SWT_CR[SLK] bits are set.

Offset: SWT_BASE + 0x0018 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	SK															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 20-7. SWT Service Register (SWT_SR)

Table 20-8. SWT_SR Field Descriptions

Field	Description
SK	Service Key.This field is the previous (or initial) service key value used in keyed service mode. If SWT_CR[KEY] is set, the next key value to be written to the SWT_SR is (17*SK+3) mod 2^{16}.

20.4 Functional Description

The SWT is a 32-bit timer designed to enable the system to recover in situations such as software getting trapped in a loop or if a bus transaction fails to terminate. It includes a a control register (SWT_CR), an interrupt register (SWT_IR), time-out register (SWT_TO), a window register (SWT_WN), a service register (SWT_SR), a counter output register (SWT_CO), and a service key register (SWT_SK).

The SWT_CR includes bits to enable the timer, set configuration options, and lock configuration of the module. The watchdog is enabled by setting the SWT_CR[WEN] bit. The reset value of the SWT_CR[WEN] bit is 1. Since the reset value of this bit is 1, the watchdog starts operation automatically after reset is released. Some devices can be configured to clear this bit automatically during the boot process.

The SWT_TO register holds the watchdog time-out period in clock cycles unless the value is less than 0x100, in which case the time-out period is set to 0x100. This time-out period is loaded into an internal 32-bit down counter when the SWT is enabled and each time a valid service operation is performed. The SWT_CR[CSL] bit selects which clock (system or oscillator) is used to drive the down counter.

NOTE

The default value of SWT_TO is updated by the BAM code during the serial download routine. See Section 9.3.3.2, Serial-Boot Mode Features, for more details.

The configuration of the SWT can be locked through use of either a soft lock or a hard lock. In either case, when locked the SWT_CR, SWT_TO, SWT_WN, and SWT_SK registers are read-only. The hard lock is enabled by setting the SWT_CR[HLK] bit which can only be cleared by a reset. The soft lock is enabled by setting the SWT_CR[SLK] bit and is cleared by writing the unlock sequence to the service register. The unlock sequence is a write of 0xC520 followed by a write of 0xD928 to the SWT_SR[WSC] field. There is no timing requirement between the two writes. The unlock sequence logic ignores service sequence writes and recognizes the 0xC520, 0xD928 sequence regardless of previous writes. The unlock sequence can be written at any time and does not require the SWT_CR[WEN] bit to be set.

When enabled, the SWT requires periodic execution of a servicing operation which consists of writing two values to the SWT_SR. Writing the proper sequence of values loads the internal down counter with the time-out period. There is no timing requirement between the two writes and the service sequence logic ignores unlock sequence writes. If the SWT_CR[KEY] bit is zero, the fixed sequence 0xA602, 0xB480 is written to the SWT_SR[WSC] field to service the watchdog. If the SWT_CR[KEY] bit is set, then two pseudorandom keys are written to the SWT_SR[WSC] field to service the watchdog. The key values are determined by the pseudorandom key generator defined in Equation 20-1. This algorithm generates a sequence of 2^{16} different key values before repeating. The state of the key generator is held in the SWT_SK register. For example, if SWT_SK[SK] is 0x0100 then the service sequence keys are 0x1103, 0x2136. In this mode, each time a valid key is written to the SWT_SR register, the SWT_SK register is updated. So, after servicing the watchdog by writing 0x1103 and then 0x2136 to the SWT_SR[WSC] field, SWT_SK[SK] is 0x2136 and the next key sequence is 0x3499, 0x7E2C.

$$SK_{n+1} = (17*SK_n+3) \bmod 2^{16}$$

Pseudorandom Key Generator ***Eqn. 20-1***

Accesses to SWT registers occur with no peripheral bus wait states. However, due to synchronization logic in the SWT design, recognition of the service sequence or configuration changes may require as many as 3 system plus 7 counter clock cycles.

If window mode is enabled (SWT_CR[WND] bit is set), the service sequence must be performed in the last part of the time-out period defined by the window register. The window is open when the down counter is less than the value in the SWT_WN register. Outside of this window, service sequence writes are invalid accesses and generate a bus error or reset depending on the value of the SWT_CR[RIA] bit. For example, if the SWT_TO register is set to 5000 and SWT_WN register is set to 1000 then the service sequence must be performed in the last 20% of the time-out period. There is a short lag in the time it takes for the window

to open due to synchronization logic in the watchdog design. This delay could be as many as 3 system plus 4 counter clock cycles.

The interrupt then reset bit (SWT_CR[ITR]) controls the action taken when a time-out occurs. If the SWT_CR[ITR] bit is not set, a reset is generated immediately on a time-out. If the SWT_CR[ITR] bit is set, an initial time-out causes the SWT to generate an interrupt and load the down counter with the time-out period. If the service sequence is not written before the second consecutive time-out, the SWT generates a system reset. The interrupt is indicated by the time-out interrupt flag (SWT_IR[TIF]). The interrupt request is cleared by writing a one to the SWT_IR[TIF] bit.

The SWT_CO register shows the value of the down counter when the watchdog is disabled. When the watchdog is enabled this register is cleared. The value shown in this register can lag behind the value in the internal counter for as many as 6 system plus 8 counter clock cycles.

The SWT_CO can be used during a software self test of the SWT. For example, the SWT can be enabled and not serviced for a fixed period of time less than the time-out value. Then the SWT can be disabled (SWT_CR[WEN] cleared) and the value of the SWT_CO read to determine if the internal down counter is working properly.

Chapter 21
System Timer Module (STM)

21.1 Overview

The System Timer Module (STM) is a 32-bit timer designed to support commonly required system and application software timing functions. The STM includes a 32-bit up counter and four 32-bit compare channels with a separate interrupt source for each channel. The counter is driven by the system clock divided by an 8-bit prescale value (1 to 256).

21.1.1 Features

The STM has the following features:

- One 32-bit up counter with 8-bit prescaler
- Four 32-bit compare channels
- Independent interrupt source for each channel
- Counter can be stopped in debug mode

21.1.2 Modes of Operation

The STM supports two device modes of operation: normal and debug. When the STM is enabled in normal mode, its counter runs continuously. In debug mode, operation of the counter is controlled by the FRZ bit in the STM_CR. If the FRZ bit is set, the counter is stopped in debug mode, otherwise it continues to run.

21.1.3 Clocking

The STM is clocked by the system clock. It can be frozen during debug by asserting the FREEZE signal.

21.1.4 Interrupts

The STM has four independent interrupt sources that are connected to the interrupt controller.

21.2 External Signal Description

The STM does not have any external interface signals.

21.3 Memory Map and Register Definition

The STM has 14 32-bit read and write access registers. The STM registers can only be accessed using 32-bit (word) accesses. Attempted references using a different size or to a reserved address generates a bus error termination.

21.3.1 Memory Map

The STM memory map is shown in Table 21-1.

Table 21-1. STM Memory Map

Offset from STM_BASE (0xFFF3_C000)	Register	Access	Reset Value[1]	Section/Page
0x0000	STM_CR – STM Control Register	R/W	0x0000_0000	21.3.2.1/21-3
0x0004	STM_CNT – STM Counter Value	R/W	0x0000_0000	21.3.2.2/21-3
0x0008–0x000F	Reserved			
0x0010	STM_CCR0 – STM Channel 0 Control Register	R/W	0x0000_0000	21.3.2.3/21-4
0x0014	STM_CIR0 – STM Channel 0 Interrupt Register	R/W	0x0000_0000	21.3.2.4/21-4
0x0018	STM_CMP0 – STM Channel 0 Compare Register	R/W	0x0000_0000	21.3.2.5/21-5
0x001C	Reserved			
0x0020	STM_CCR1 – STM Channel 1 Control Register	R/W	0x0000_0000	21.3.2.3/21-4
0x0024	STM_CIR1 – STM Channel 1 Interrupt Register	R/W	0x0000_0000	21.3.2.4/21-4
0x0028	STM_CMP1 – STM Channel 1 Compare Register	R/W	0x0000_0000	21.3.2.5/21-5
0x002C	Reserved			
0x0030	STM_CCR2 – STM Channel 2 Control Register	R/W	0x0000_0000	21.3.2.3/21-4
0x0034	STM_CIR2 – STM Channel 2 Interrupt Register	R/W	0x0000_0000	21.3.2.4/21-4
0x0038	STM_CMP2 – STM Channel 2 Compare Register	R/W	0x0000_0000	21.3.2.5/21-5
0x003C	Reserved			
0x0040	STM_CCR3 – STM Channel 3 Control Register	R/W	0x0000_0000	21.3.2.3/21-4
0x0044	STM_CIR3 – STM Channel 3 Interrupt Register	R/W	0x0000_0000	21.3.2.4/21-4
0x0048	STM_CMP3 – STM Channel 3 Compare Register	R/W	0x0000_0000	21.3.2.5/21-5
0x004C–0x3FFF	Reserved			

[1] See register definition.

21.3.2 Register Descriptions

The following sections detail the individual registers within the STM.

21.3.2.1 STM Control Register (STM_CR)

The STM Control Register (STM_CR) includes the prescale value, freeze control, and timer enable bits.

Offset: STM_BASE + 0x0000 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CPS								0	0	0	0	0	0	FRZ	TEN
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 21-1. STM Control Register (STM_CR)

Table 21-2. STM_CR Field Descriptions

Field	Description
CPS	Counter Prescaler. Selects the clock divide value for the prescaler (1–256). 0x00 Divide system clock by 1. 0x01 Divide system clock by 2. ... 0xFF Divide system clock by 256.
FRZ	Freeze. Allows the timer counter to be stopped when the device enters debug mode. 0 STM counter continues to run in debug mode. 1 STM counter is stopped in debug mode.
TEN	Timer Counter Enabled. 0 Counter is disabled. 1 Counter is enabled.

21.3.2.2 STM Count Register (STM_CNT)

The STM Count Register (STM_CNT) holds the timer count value.

Offset STM_BASE + 0x0004 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CNT																															
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 21-2. STM Count Register (STM_CNT)

Table 21-3. STM_CNT Field Descriptions

Field	Description
CNT	Timer count value used as the time base for all channels. When enabled, the counter increments at the rate of the system clock divided by the prescale value.

21.3.2.3 STM Channel Control Register (STM_CCR*n*)

The STM Channel Control Register (STM_CCR*n*) is used to enable and service channel *n* of the timer.

Offset STM_CCR0: STM_BASE + 0x0010 Access: User read/write
STM_CCR1: STM_BASE + 0x0020
STM_CCR2: STM_BASE + 0x0030
STM_CCR3: STM_BASE + 0x0040

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	CEN
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 21-3. STM Channel Status and Control Register (STM_CCR*n*)

Table 21-4. STM_CCR*n* Field Descriptions

Field	Description
CEN	Channel Enable. 0 The channel is disabled. 1 The channel is enabled.

21.3.2.4 STM Channel Interrupt Register (STM_CIR*n*)

The STM Channel Interrupt Register (STM_CIR*n*) is used to enable and service channel *n* of the timer.

Offset: STM_CIR0: STM_BASE + 0x0014 Access: User read/write
STM_CIR1: STM_BASE + 0x0024
STM_CIR2: STM_BASE + 0x0034
STM_CIR3: STM_BASE + 0x0044

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	CIF
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 21-4. STM Channel Interrupt Register (STM_CIR*n*)

Table 21-5. STM_CIR*n* Field Descriptions

Field	Description
CIF	Channel Interrupt Flag. The flag and interrupt are cleared by writing a 1 to this bit. Writing a 0 has no effect. 0 No interrupt request. 1 Interrupt request due to a match on the channel.

21.3.2.5 STM Channel Compare Register (STM_CMP*n*)

The STM channel compare register (STM_CMP*n*) holds the compare value for channel *n*.

Offset: STM_CMP0: STM_BASE + 0x0018 Access: User read/write
STM_CMP1: STM_BASE + 0x0028
STM_CMP2: STM_BASE + 0x0038
STM_CMP3: STM_BASE + 0x0048

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CMP																															
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 21-5. STM Channel Compare Register (STM_CMP*n*)

Table 21-6. STM_CMP*n* Register Field Descriptions

Field	Description
CMP	Compare value for channel *n*. If the STM_CCR*n*[CEN] bit is set and the STM_CMP*n* register matches the STM_CNT*n* register, a channel interrupt request is generated and the STM_CIR*n*[CIF] bit is set.

21.4 Functional Description

The System Timer Module (STM) is a 32-bit timer designed to support commonly required system and application software timing functions. The STM includes a 32-bit up counter and four 32-bit compare channels with a separate interrupt source for each channel.

The STM has one 32-bit up counter (STM_CNT) that is used as the time base for all channels. When enabled, the counter increments at the system clock frequency divided by a prescale value. The STM_CR[CPS] field sets the divider to any value in the range from 1 to 256. The counter is enabled with the STM_CR[TEN] bit. When enabled in normal mode the counter continuously increments. When enabled in debug mode the counter operation is controlled by the STM_CR[FRZ] bit. When the STM_CR[FRZ] bit is set, the counter is stopped in debug mode, otherwise it continues to run in debug mode. The counter rolls over at 0xFFFF_FFFF to 0x0000_0000 with no restrictions at this boundary.

The STM has four identical compare channels. Each channel includes a channel control register (STM_CCR*n*), a channel interrupt register (STM_CIR*n*) and a channel compare register (STM_CMP*n*). The channel is enabled by setting the STM_CCR*n*[CEN] bit. When enabled, the channel sets the STM_CIR*n*[CIF] bit and generate an interrupt request when the channel compare register matches the timer counter. The interrupt request is cleared by writing a 1 to the STM_CIR*n*[CIF] bit. A write of 0 to the STM_CIR*n*[CIF] bit has no effect.

Chapter 22
Periodic Interrupt Timer (PIT)

22.1 Introduction

The Periodic Interrupt Timer (PIT) is an array of timers that can be used to initiate interrupts and trigger DMA channels.

22.1.1 Block Diagram

A simplified block diagram of the PIT illustrates the functionality and interdependence of major blocks (see Figure 22-1).

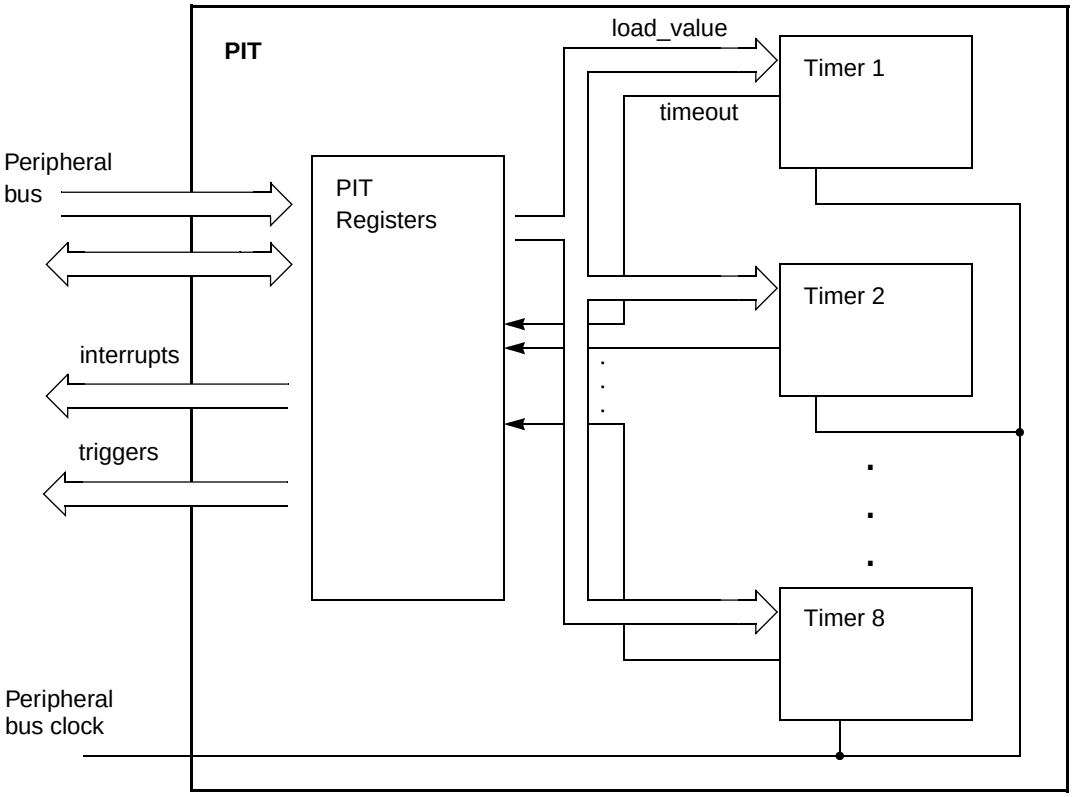


Figure 22-1. PIT Block Diagram

22.1.2 Features

The PIT has these major features:

- Eight 32-bit timers generating DMA trigger pulses
- All timers can be configured to generate interrupts instead of triggers
- Timer 3 can be the source of an ADC trigger input via SIU configuration
- Timers share one common core clock
- Independent timeout periods for each timer

Table 22-1. Timer Features

Timer	Interrupt Vector Number	DMA Trigger	ADC Trigger	CTU Trigger
1	149	X		
2	150	X		
3	151	X	X	
4	152	X		X
5	153	X		
6	154	X		
7	155	X		
8	156	X		

22.1.3 Modes of Operation

There are two main operating modes of PIT: run mode and halt mode. In run mode, bit 7 = 0 in the SIU_HLT0 register and all functional parts of the PIT module are running. In halt mode, bit 7 = 1 in the SIU_HLT0 register, and the clock to the PIT module is disabled, halting the module.

22.2 Signal Description

22.2.1 External Signal Description

The PIT module has no external signals.

22.3 Memory Map and Registers

This section provides a detailed description of all PIT registers.

22.3.1 Module Memory Map

The PIT memory map is shown in Table 22-2. The address of each register is given as an offset to the PIT base address. Registers are listed in address order, identified by complete name and mnemonic, with the type of accesses allowed.

Table 22-2. PIT Memory Map

Offset from PIT_BASE (0xFFFE_0000)	Register	Access	Reset Value	Section/Page
0x0000	PITMCR—PIT Module Control Register	R/W[1]	0x0000_0000	22.3.2.1/22-4
0x0004–0x00FF	Reserved			
Timer Channel 1				
0x0100	LDVAL1—Timer 1 Load Value Register	R/W	0x0000_0000	22.3.2.2/22-5
0x0104	CVAL1—Timer 1 Current Value Register	R/W	0x0000_0000	22.3.2.3/22-5
0x0108	TCTRL1—Timer 1 Control Register	R/W[1]	0x0000_0000	22.3.2.4/22-6
0x010C	TFLG1—Timer 1 Flag Register	R/W[1]	0x0000_0000	22.3.2.5/22-7
Timer Channel 2				
0x0110	LDVAL2—Timer 2 Load Value Register	R/W	0x0000_0000	22.3.2.2/22-5
0x0114	CVAL2—Timer 2 Current Value Register	R/W	0x0000_0000	22.3.2.3/22-5
0x0118	TCTRL2—Timer 2 Control Register	R/W[1]	0x0000_0000	22.3.2.4/22-6
0x011C	TFLG2—Timer 2 Flag Register	R/W[1]	0x0000_0000	22.3.2.5/22-7
Timer Channel 3				
0x0120	LDVAL3—Timer 3 Load Value Register	R/W	0x0000_0000	22.3.2.2/22-5
0x0124	CVAL3—Timer 3 Current Value Register	R/W	0x0000_0000	22.3.2.3/22-5
0x0128	TCTRL3—Timer 3 Control Register	R/W[1]	0x0000_0000	22.3.2.4/22-6
0x012C	TFLG3—Timer 3 Flag Register	R/W[1]	0x0000_0000	22.3.2.5/22-7
Timer Channel 4				
0x0130	LDVAL4—Timer 4 Load Value Register	R/W	0x0000_0000	22.3.2.2/22-5
0x0134	CVAL4—Timer 4 Current Value Register	R/W	0x0000_0000	22.3.2.3/22-5
0x0138	TCTRL4—Timer 4 Control Register	R/W[1]	0x0000_0000	22.3.2.4/22-6
0x013C	TFLG4—Timer 4 Flag Register	R/W[1]	0x0000_0000	22.3.2.5/22-7
Timer Channel 5				
0x0140	LDVAL5—Timer 5 Load Value Register	R/W	0x0000_0000	22.3.2.2/22-5
0x0144	CVAL5—Timer 5 Current Value Register	R/W	0x0000_0000	22.3.2.3/22-5
0x0148	TCTRL5—Timer 5 Control Register	R/W[1]	0x0000_0000	22.3.2.4/22-6
0x014C	TFLG5—Timer 5 Flag Register	R/W[1]	0x0000_0000	22.3.2.5/22-7
Timer Channel 6				
0x0150	LDVAL6—Timer 6 Load Value Register	R/W	0x0000_0000	22.3.2.2/22-5
0x0154	CVAL6—Timer 6 Current Value Register	R/W	0x0000_0000	22.3.2.3/22-5
0x0158	TCTRL6—Timer 6 Control Register	R/W[1]	0x0000_0000	22.3.2.4/22-6
0x015C	TFLG6—Timer 6 Flag Register	R/W[1]	0x0000_0000	22.3.2.5/22-7

Table 22-2. PIT Memory Map (continued)

Offset from PIT_BASE (0xFFFE_0000)	Register	Access	Reset Value	Section/Page
Timer Channel 7				
0x0160	LDVAL7—Timer 7 Load Value Register	R/W	0x0000_0000	22.3.2.2/22-5
0x0164	CVAL7—Timer 7 Current Value Register	R/W	0x0000_0000	22.3.2.3/22-5
0x0168	TCTRL7—Timer 7 Control Register	R/W[1]	0x0000_0000	22.3.2.4/22-6
0x016C	TFLG7—Timer 7 Flag Register	R/W[1]	0x0000_0000	22.3.2.5/22-7
Timer Channel 8				
0x0170	LDVAL8—Timer 8 Load Value Register	R/W	0x0000_0000	22.3.2.2/22-5
0x0174	CVAL8—Timer 8 Current Value Register	R/W	0x0000_0000	22.3.2.3/22-5
0x0178	TCTRL8—Timer 8 Control Register	R/W[1]	0x0000_0000	22.3.2.4/22-6
0x017C	TFLG8—Timer 8 Flag Register	R/W[1]	0x0000_0000	22.3.2.5/22-7
0x0180–0x03FF	Reserved			

1 Some bits are read-only.

22.3.2 Register Descriptions

This section lists the PIT registers and describes the registers and their bit fields.

22.3.2.1 PIT Module Control Register (PITMCR)

This register controls whether the timer clocks should be enabled and whether the timers should run in debug mode.

Offset: PIT_BASE + 0x0000 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	MDIS	FRZ
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0

Figure 22-2. PIT Module Control Register (PITMCR)

Table 22-3. PITMCR Field Descriptions

Field	Description
MDIS	Module Disable. This is used to disable the module clock. This bit should be enabled before any other setup is done. 0 Clock for PIT Timers is enabled. 1 Clock for PIT Timers is disabled (default).
FRZ	Freeze. Allows the timers to be stopped when the device enters debug mode. 0 Timers continue to run in debug mode. 1 Timers are stopped in debug mode.

22.3.2.2 Timer *n* Load Value Register (LDVAL*n*)

These registers select the timeout period for the timer interrupts. Changes to this value are visible immediately.

Offset: Channel_base + 0x0000 Access: User read/write

LDVAL1 = 0x0100 LDVAL5 = 0x0140
LDVAL2 = 0x0110 LDVAL6 = 0x0150
LDVAL3 = 0x0120 LDVAL7 = 0x0160
LDVAL4 = 0x0130 LDVAL8 = 0x0170

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	TSV31	TSV30	TSV29	TSV28	TSV27	TSV26	TSV25	TSV24	TSV23	TSV22	TSV21	TSV20	TSV19	TSV18	TSV17	TSV16
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	TSV15	TSV14	TSV13	TSV12	TSV11	TSV10	TSV9	TSV8	TSV7	TSV6	TSV5	TSV4	TSV3	TSV2	TSV1	TSV0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 22-3. Timer *n* Load Value Register (LDVAL*n*)

Table 22-4. LDVAL*n* Field Descriptions

Field	Description
TSV*n*	Time Start Value Bits. These bits set the timer start value. The timer counts down until it reaches 0. Then it generates an interrupt and loads this register value again. Writing a new value to this register does not restart the timer. Instead, the value is reloaded once the timer expires. To stop the current cycle and start a timer period with the new value, the timer must be disabled and enabled again (see Figure 22-8).

22.3.2.3 Timer *n* Current Value Register (CVAL*n*)

These registers indicate the current timer position.

Offset: Channel_base + 0x0004 Access: User read/write

LDVAL1 = 0x0104 LDVAL5 = 0x0144
LDVAL2 = 0x0114 LDVAL6 = 0x0154
LDVAL3 = 0x0124 LDVAL7 = 0x0164
LDVAL4 = 0x0134 LDVAL8 = 0x0174

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	TVL31	TVL30	TVL29	TVL28	TVL27	TVL26	TVL25	TVL24	TVL23	TVL22	TVL21	TVL20	TVL19	TVL18	TVL17	TVL16
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	TVL15	TVL14	TVL13	TVL12	TVL11	TVL10	TVL9	TVL8	TVL7	TVL6	TVL5	TVL4	TVL3	TVL2	TVL1	TVL0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 22-4. Timer *n* Current Value Register (CVAL*n*)

Table 22-5. CVAL*n* Field Descriptions

Field	Description
TVL*n*	Current Timer Value. These bits represent the current timer value. Note that the timer uses a down counter. **Note:** The timer values are frozen in Debug mode if the FRZ bit is set in the PIT Module Control Register (see Figure 22-2).

22.3.2.4 Timer *n* Control Register (TCTRL*n*)

These registers contain the control bits for each timer.

Offset: Channel_base + 0x0008 Access: User read/write

LDVAL1 = 0x0108 LDVAL5 = 0x0148
LDVAL2 = 0x0118 LDVAL6 = 0x0158
LDVAL3 = 0x0128 LDVAL7 = 0x0168
LDVAL4 = 0x0138 LDVAL8 = 0x0178

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	TIE	TEN
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 22-5. Timer *n* Control Register (TCTRL*n*)

Table 22-6. TCTRL*n* Field Descriptions

Field	Description
TIE	Timer Interrupt Enable Bit. 0 Interrupt requests from Timer *n* are disabled. 1 Interrupt will be requested whenever TIF is set. When an interrupt is pending (TIF set), enabling the interrupt will immediately cause an interrupt event. To avoid this, the associated TIF flag must be cleared first.
TEN	Timer Enable Bit. 0 Timer is disabled. 1 Timer is active.

22.3.2.5 Timer *n* Flag Register (TFLG*n*)

These registers hold the PIT interrupt flags.

Offset: Channel_base + 0x000C — Access: User read/write

LDVAL1 = 0x010C, LDVAL2 = 0x011C, LDVAL3 = 0x012C, LDVAL4 = 0x013C, LDVAL5 = 0x014C, LDVAL6 = 0x015C, LDVAL7 = 0x016C, LDVAL8 = 0x017C

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	TIF
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0

Figure 22-6. Timer *n* Flag Register (TFLG*n*)

Table 22-7. TFLG*n* Field Descriptions

Field	Description
TIF	Time Interrupt Flag. TIF is set to 1 at the end of the timer period.This flag can be cleared only by writing it with a 1. Writing a 0 has no effect. If enabled (TIE = 1), TIF causes an interrupt request. 0 Time-out has not yet occurred. 1 Time-out has occurred.

22.4 Functional Description

The PIT block has eight timers for general-purpose use. Each timer can be used to generate trigger pulses as well as to generate interrupts. Each interrupt is available on a separate interrupt line.

22.4.1 Timers

The timers generate triggers at periodic intervals, when enabled. They load their start values, as specified in their LDVAL*n* registers, then count down until they reach 0. Then they load their respective start value again. Each time a timer reaches 0, it will generate a trigger pulse, and set the interrupt flag.

All interrupts can be enabled or masked (by setting the TIE bits in the TCTRL*n* registers). A new interrupt can be generated only after the previous one is cleared.

If desired, the current counter value of the timer can be read via the CVAL*n* registers.

The counter period can be restarted by first disabling and then re-enabling the timer, using the TEN bit (see Figure 22-7).

The counter period of a running timer can be modified, by first disabling the timer, setting a new load value and then enabling the timer again (see Figure 22-8).

It is also possible to change the counter period without restarting the timer by writing the LDVAL*n* register with the new load value. This value will then be loaded after the next trigger event (see Figure 22-9).

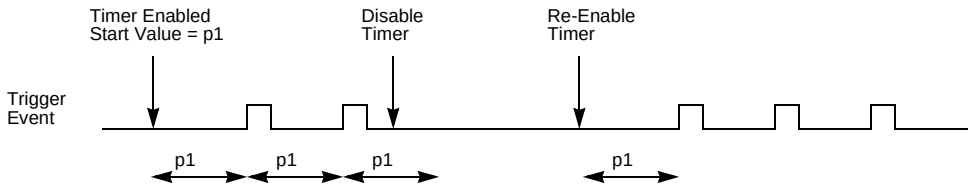


Figure 22-7. Stopping and Starting a Timer

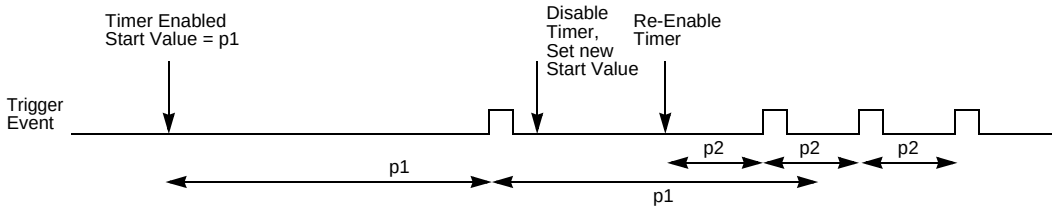


Figure 22-8. Modifying Running Timer Period

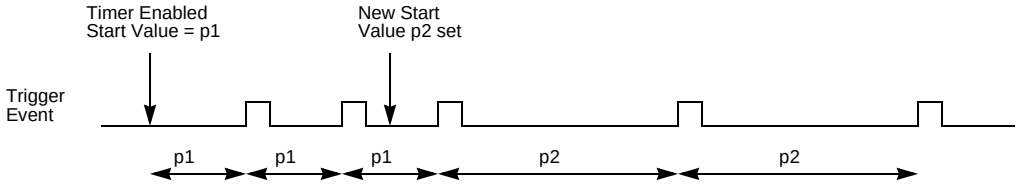


Figure 22-9. Dynamically Setting a New Load Value

22.4.2 Debug Mode

In debug mode, the timers are frozen. This is intended to aid software development, allowing the developer to halt the processor, investigate the current state of the system (e.g., the timer values), and then continue the operation.

22.4.3 Interrupts

All of the timers support interrupt generation. SeeChapter 10, Interrupts and Interrupt Controller (INTC), for related vector addresses and priorities.

Timer interrupts can be disabled by setting the TIE bits to 0. The timer interrupt flags (TIF) are set to 1 when a timeout occurs on the associated timer, and are cleared to 0 by writing a 1 to that TIF bit.

22.5 Initialization and Application Information

22.5.1 Example Configuration

In the example configuration:

- The PIT clock has a frequency of 50 MHz
- Timer 1 creates an interrupt every 5.12 ms
- Timer 8 creates a trigger event every 30 ms.

First the PIT module needs to be activated by writing a 0 to the MDIS bit in the PITMCR register.

The 50 MHz clock frequency equates to a clock period of 20 ns and the 10 MHz frequency equates to a clock period of 100 ns. Timer 1 needs to trigger every 5.12 ms/20 ns = 256,000 cycles, and Timer 8 needs to trigger every 30 ms/20 ns = 1,500,000 cycles. The value for the LDVAL*n* register trigger would be calculated as (period / clock period) – 1.

This means that LDVAL1 will be written with 0x0003_E7FF, and LDVAL8 with 0x0016_E35F.

Enable the interrupt for Timer 1 is doing the following:

1. Clear TIF in TFLG1. (If TIF is set, an interrupt occurs immediately when Timer 1 is enabled.)
2. Set TIE in the TCTRL1 register.
3. Start the timer by writing a 1 to bit TEN in the TCTRL1 register.

Timer 8 is used for triggering only. Timer 8 is started by writing a 1 to bit TEN in the TCTRL8 register.

The following example code matches the described setup:

```
// turn on PIT
PIT_REG_P->pit_CTRL = 0x00;

// Timer 1
PIT_REG_P->pit_LDVAL1 = 0x0003E7FF; // setup Timer 1 for 256000 cycles
PIT_REG_P->pit_TCTRL1 |= 1<<1; // enable Timer 1 interrupts
PIT_REG_P->pit_EN |= 1<<1; // start timer 1

// Timer 8
PIT_REG_P->pit_LDVAL8 = 0x0016E35F; // setup timer 8 for 1500000 cycles
// timer 8 can’t generate interrupts -> no settings needed for trigger
PIT_REG_P->pit_EN |= 1<<8; // start timer 8
```

Chapter 23
DMA Channel Multiplexer (DMA_MUX)

23.1 Introduction

The DMA channel multiplexer (DMA_MUX) module allows for software selection of 32 out of 59 possible DMA sources. As many as 55 of these DMA sources are requests from peripherals, but four of the peripheral sources are reserved and behave as always disabled sources. Four sources are always enabled and generate a DMA request as soon as that source is selected. One source (the default for all channels) is always disabled.

23.1.1 Block Diagram

A simplified block diagram of the DMA_MUX is shown in Figure 23-1.

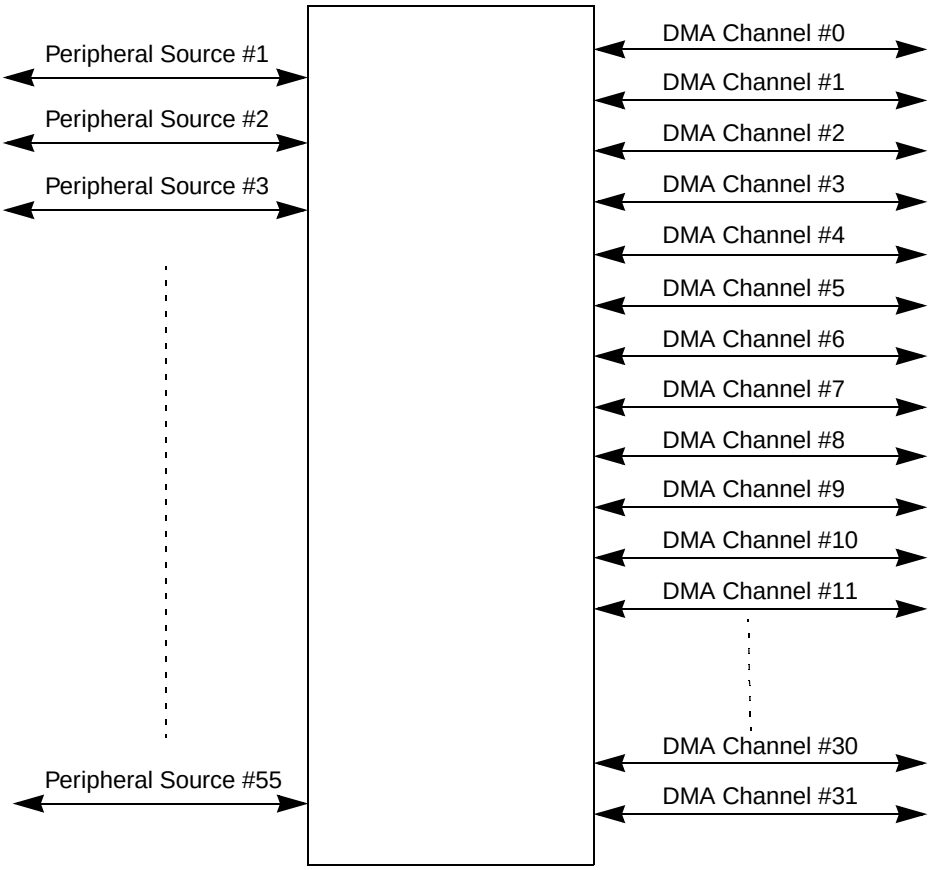


Figure 23-1. DMA_MUX Block Diagram

23.1.2 Features

The DMA_MUX has these major features:

- 32 independently selectable DMA channel routers
  - Four channels with normal or periodic triggering capability
  - 24 channels with normal operation only
  - Each channel router can be assigned to 1 of 55 possible peripheral DMA sources, eight always enabled sources, or one always disabled source.

23.1.3 Modes of Operation

DMA channels 0–7 may be used in the following modes, but channels 8–31 may only be configured to disabled or normal mode.

- Disabled mode

  In this mode, the DMA channel is disabled. Because disabling and enabling of DMA channels is done primarily via the DMA registers, this mode is used mainly as the reset state for a DMA channel in the DMA channel mux. It may also be used to temporarily suspend a DMA channel while reconfiguration of the system takes place (changing the period of a DMA trigger, for example).
- Normal mode

  In this mode, a DMA source (such as SCI transmit or SCI receive for example) is routed directly to the specified DMA channel. The operation of the DMA_MUX in this mode is completely transparent to the system.
- Periodic trigger mode

  In this mode, a DMA source may only request a DMA transfer (such as when a transmit buffer becomes empty or a receive buffer becomes full) periodically. Configuration of the period is done in the registers of the periodic interrupt timer.

23.2 External Signal Description

The DMA_MUX has no external signals.

23.3 Memory Map and Registers

This section provides a detailed description of all DMA_MUX registers.

23.3.1 Module Memory Map

The DMA_MUX memory map is shown in Table 23-1. The address of each register is given as an offset to the DMA_MUX base address. Registers are listed in address order, identified by complete name and mnemonic, and list the type of accesses allowed.

All registers are accessible via 8-bit, 16-bit, or 32-bit accesses. However, 16-bit accesses must be aligned to 16-bit boundaries and 32-bit accesses must be aligned to 32-bit boundaries. As an example,

CHCONFIG0 through CHCONFIG4 are accessible by a 32-bit READ/WRITE to address DMA_MUX_BASE + 0x00, but performing a 32-bit access to address DMA_MUX_BASE + 0x01 is illegal.

Table 23-1. DMA_MUX Memory Map

Offset from DMA_MUX_BASE (0xFFFD_C000)	Register	Access	Reset Value	Section/Page
0x0000	CHCONFIG0—Channel #0 configuration	R/W	0x00	23.3.2.1/23-4
0x0001	CHCONFIG1—Channel #1 configuration	R/W	0x00	23.3.2.1/23-4
0x0002	CHCONFIG2—Channel #2 configuration	R/W	0x00	23.3.2.1/23-4
0x0003	CHCONFIG3—Channel #3 configuration	R/W	0x00	23.3.2.1/23-4
0x0004	CHCONFIG4—Channel #4 configuration	R/W	0x00	23.3.2.1/23-4
0x0005	CHCONFIG5—Channel #5 configuration	R/W	0x00	23.3.2.1/23-4
0x0006	CHCONFIG6—Channel #6 configuration	R/W	0x00	23.3.2.1/23-4
0x0007	CHCONFIG7—Channel #7 configuration	R/W	0x00	23.3.2.1/23-4
0x0008	CHCONFIG8—Channel #8 configuration	R/W	0x00	23.3.2.1/23-4
0x0009	CHCONFIG9—Channel #9 configuration	R/W	0x00	23.3.2.1/23-4
0x000A	CHCONFIG10—Channel #10 configuration	R/W	0x00	23.3.2.1/23-4
0x000B	CHCONFIG11—Channel #11 configuration	R/W	0x00	23.3.2.1/23-4
0x000C	CHCONFIG12—Channel #12 configuration	R/W	0x00	23.3.2.1/23-4
0x000D	CHCONFIG13—Channel #13 configuration	R/W	0x00	23.3.2.1/23-4
0x000E	CHCONFIG14—Channel #14 configuration	R/W	0x00	23.3.2.1/23-4
0x000F	CHCONFIG15—Channel #15 configuration	R/W	0x00	23.3.2.1/23-4
0x0010	CHCONFIG16—Channel #16 configuration	R/W	0x00	23.3.2.1/23-4
0x0011	CHCONFIG17—Channel #17 configuration	R/W	0x00	23.3.2.1/23-4
0x0012	CHCONFIG18—Channel #18 configuration	R/W	0x00	23.3.2.1/23-4
0x0013	CHCONFIG19—Channel #19 configuration	R/W	0x00	23.3.2.1/23-4
0x0014	CHCONFIG20—Channel #20 configuration	R/W	0x00	23.3.2.1/23-4
0x0015	CHCONFIG21—Channel #21 configuration	R/W	0x00	23.3.2.1/23-4
0x0016	CHCONFIG22—Channel #22 configuration	R/W	0x00	23.3.2.1/23-4
0x0017	CHCONFIG23—Channel #23 configuration	R/W	0x00	23.3.2.1/23-4
0x0018	CHCONFIG24—Channel #24 configuration	R/W	0x00	23.3.2.1/23-4
0x0019	CHCONFIG25—Channel #25 configuration	R/W	0x00	23.3.2.1/23-4
0x001A	CHCONFIG26—Channel #26 configuration	R/W	0x00	23.3.2.1/23-4
0x001B	CHCONFIG27—Channel #27 configuration	R/W	0x00	23.3.2.1/23-4
0x001C	CHCONFIG28—Channel #28 configuration	R/W	0x00	23.3.2.1/23-4

Table 23-1. DMA_MUX Memory Map (continued)

Offset from DMA_MUX_BASE (0xFFFD_C000)	Register	Access	Reset Value	Section/Page
0x001D	CHCONFIG29—Channel #29 configuration	R/W	0x00	23.3.2.1/23-4
0x001E	CHCONFIG30—Channel #30 configuration	R/W	0x00	23.3.2.1/23-4
0x001F	CHCONFIG31—Channel #31 configuration	R/W	0x00	23.3.2.1/23-4

23.3.2 Register Descriptions

This section lists the DMA_MUX registers in address order and describes the registers and their bit fields.

23.3.2.1 Channel Configuration Registers (CHCONFIG*x*)

Each of the 32 DMA channels can be independently enabled/disabled and associated with one of the 59 DMA sources in the system.

Offset: DMA_MUX_BASE + *x* – 1 Access: User read/write

	0	1	2	3	4	5	6	7
R	ENBL	TRIG	SOURCE					
W								
Reset	0	0	0	0	0	0	0	0

Figure 23-2. Channel Configuration Registers (CHCONFIG*x*)

Table 23-2. CHCONFIG*x* Field Descriptions

Field	Description
ENBL	DMA Channel Enable. ENBL enables the DMA channel. 0 DMA channel is disabled. This mode is primarily used during configuration of the DMA_MUX. The DMA has separate channel enables/disables, which should be used to disable or re-configure a DMA channel. 1 DMA channel is enabled.
TRIG	DMA Channel Trigger Enable (channels 0–7 only). TRIG enables the periodic trigger capability for the DMA channel 0 Triggering is disabled. If triggering is disabled and the ENBL bit is set, the DMA channel routes the specified source to the DMA channel. 1 Triggering is enabled.
SOURCE	DMA Channel Source. SOURCE specifies which DMA source, if any, is routed to a particular DMA channel, according to Table 23-4.

Table 23-3. Channel and Trigger Enabling

ENBL	TRIG	Function	Mode
0	X	DMA channel is disabled	Disabled mode
1	0	DMA channel is enabled with no triggering (transparent)	Normal mode
1	1	DMA channel is enabled with triggering	Periodic trigger mode

NOTE

Setting multiple CHCONFIG registers with the same DMA source value results in unpredictable behavior.

Table 23-4. DMA Source Configuration

DMA Request	DMA_MUX Source Input Number	DMA Source	Description
Channel disabled[1]	0x00	Channel disabled	Channel disabled
Reserved	0x01	Reserved	Reserved
SCI_A_COMBTX	0x02	SCI_A.SCISR1[TDRE] || SCI_A.SCISR1[TC] || SCI_A.LINSTAT1[TXRDY]	SCI_A combined DMA request of the transmit data register empty, transmit complete, and LIN transmit data ready DMA requests
SCI_A_COMBRX	0x03	SCI_A.SCISR1[RDRF] || SCI_A.LINSTAT1[RXRDY]	SCI_A combined DMA request of the receive data register full and LIN receive data ready DMA requests
SCI_B_COMBTX	0x04	SCI_B.SCISR1[TDRE] || SCI_B.SCISR1[TC] || SCI_B.LINSTAT1[TXRDY]	SCI_B combined DMA request of the transmit data register empty, transmit complete, and LIN transmit data ready DMA requests
SCI_B_COMBRX	0x05	SCI_B.SCISR1[RDRF] || SCI_B.LINSTAT1[RXRDY]	SCI_B combined DMA request of the receive data register full and LIN receive data ready DMA requests
SCI_C_COMBTX	0x06	SCI_C.SCISR1[TDRE] || SCI_C.SCISR1[TC] || SCI_C.LINSTAT1[TXRDY]	SCI_C combined DMA request of the transmit data register empty, transmit complete, and LIN transmit data ready DMA requests
SCI_C_COMBRX	0x07	SCI_C.SCISR1[RDRF] || SCI_C.LINSTAT1[RXRDY]	SCI_C combined DMA request of the receive data register full and LIN receive data ready DMA requests
SCI_D_COMBTX	0x08	SCI_D.SCISR1[TDRE] || SCI_D.SCISR1[TC] || SCI_D.LINSTAT1[TXRDY]	SCI_D combined DMA request of the transmit data register empty, transmit complete, and LIN transmit data ready DMA requests
SCI_D_COMBRX	0x09	SCI_D.SCISR1[RDRF] || SCI_D.LINSTAT1[RXRDY]	SCI_D combined DMA request of the receive data register full and LIN receive data ready DMA requests
SCI_E_COMBTX	0x0A	SCI_E.SCISR1[TDRE] || SCI_E.E.SCISR1[TC] || SCI_E.LINSTAT1[TXRDY]	SCI_E combined DMA request of the transmit data register empty, transmit complete, and LIN transmit data ready DMA requests
SCI_E_COMBRX	0x0B	SCI_E.SCISR1[RDRF] || SCI_E.LINSTAT1[RXRDY]	SCI_E combined DMA request of the receive data register full and LIN receive data ready DMA requests
SCI_F_COMBTX	0x0C	SCI_F.SCISR1[TDRE] || SCI_F.SCISR1[TC] || SCI_F.LINSTAT1[TXRDY]	SCI_F combined DMA request of the transmit data register empty, transmit complete, and LIN transmit data ready DMA requests
SCI_F_COMBRX	0x0D	SCI_F.SCISR1[RDRF] || SCI_F.LINSTAT1[RXRDY]	SCI_F combined DMA request of the receive data register full and LIN receive data ready DMA requests

Table 23-4. DMA Source Configuration (continued)

DMA Request	DMA_MUX Source Input Number	DMA Source	Description
SCI_G_COMBTX	0x0E	SCI_G.SCISR1[TDRE] || SCI_G.SCISR1[TC] || SCI_G.LINSTAT1[TXRDY]	SCI_G combined DMA request of the transmit data register empty, transmit complete, and LIN transmit data ready DMA requests
SCI_G_COMBRX	0x0F	SCI_G.SCISR1[RDRF] || SCI_G.LINSTAT1[RXRDY]	SCI_G combined DMA request of the receive data register full and LIN receive data ready DMA requests
SCI_H_COMBTX	0x10	SCI_H.SCISR1[TDRE] || SCI_H.SCISR1[TC] || SCI_H.LINSTAT1[TXRDY]	SCI_H combined DMA request of the transmit data register empty, transmit complete, and LIN transmit data ready DMA requests
SCI_H_COMBRX	0x11	SCI_H.SCISR1[RDRF] || SCI_H.LINSTAT1[RXRDY]	SCI_H combined DMA request of the receive data register full and LIN receive data ready DMA requests
DSPI_A_SR_TFFF	0x12	DSPI_A.DSPI_SR[TFFF]	DSPI_A transmit FIFO fill flag
DSPI_A_SR_RFDF	0x13	DSPI_A.DSPI_SR[RFDF]	DSPI_A receive FIFO drain flag
DSPI_B_SR_TFFF	0x14	DSPI_B.DSPI_SR[TFFF]	DSPI_B transmit FIFO fill flag
DSPI_B_SR_RFDF	0x15	DSPI_B.DSPI_SR[RFDF]	DSPI_B receive FIFO drain flag
DSPI_E_SR_TFFF	0x16	DSPI_C.DSPI_SR[TFFF]	DSPI_C transmit FIFO fill flag
DSPI_E_SR_RFDF	0x17	DSPI_C.DSPI_SR[RFDF]	DSPI_C receive FIFO drain flag
DSPI_F_SR_TFFF	0x18	DSPI_D.DSPI_SR[TFFF]	DSPI_D transmit FIFO fill flag
DSPI_F_SR_RFDF	0x19	DSPI_D.DSPI_SR[RFDF]	DSPI_D receive FIFO drain flag
eMIOS200_FLAG_F0	0x1A	eMIOS200.eMIOS200FLAG[F0]	eMIOS200 channel 0 flag
eMIOS200_FLAG_F1	0x1B	eMIOS200.eMIOS200FLAG[F1]	eMIOS200 channel 1 flag
eMIOS200_FLAG_F2	0x1C	eMIOS200.eMIOS200FLAG[F2]	eMIOS200 channel 2 fag
eMIOS200_FLAG_F3	0x1D	eMIOS200.eMIOS200FLAG[F3]	eMIOS200 channel 3 flag
eMIOS200_FLAG_F4	0x1E	eMIOS200.eMIOS200FLAG[F4]	eMIOS200 channel 4 flag
eMIOS200_FLAG_F5	0x1F	eMIOS200.eMIOS200FLAG[F5]	eMIOS200 channel 5 flag
eMIOS200_FLAG_F6	0x20	eMIOS200.eMIOS200FLAG[F6]	eMIOS200 channel 6 flag
eMIOS200_FLAG_F7	0x21	eMIOS200.eMIOS200FLAG[F7]	eMIOS200 channel 7 flag
eMIOS200_FLAG_F8	0x22	eMIOS200.eMIOS200FLAG[F8]	eMIOS200 channel 8 flag
eMIOS200_FLAG_F9	0x23	eMIOS200.eMIOS200FLAG[F9]	eMIOS200 channel 9 flag
eMIOS200_FLAG_F10	0x24	eMIOS200.eMIOS200FLAG[F10]	eMIOS200 channel 10 flag
eMIOS200_FLAG_F11	0x25	eMIOS200.eMIOS200FLAG[F11]	eMIOS200 channel 11 flag
eMIOS200_FLAG_F12	0x26	eMIOS200.eMIOS200FLAG[F12]	eMIOS200 channel 12 flag
eMIOS200_FLAG_F13	0x27	eMIOS200.eMIOS200FLAG[F13]	eMIOS200 channel 13 flag
eMIOS200_FLAG_F14	0x28	eMIOS200.eMIOS200FLAG[F14]	eMIOS200 channel 14 flag

Table 23-4. DMA Source Configuration (continued)

DMA Request	DMA_MUX Source Input Number	DMA Source	Description
eMIOS200_FLAG_F15	0x29	eMIOS200.eMIOS200FLAG[F15]	eMIOS200 channel 15 flag
I^2C_A_TX	0x2A	I^2C_A.TX_REQ	I^2C_A transmit
I^2C_A_RX	0x2B	I^2C_A.RX_REQ	I^2C_A receive
I^2C_B_TX	0x2C	I^2C_B.TX_REQ	I^2C_B transmit
I^2C_B_RX	0x2D	I^2C_B.RX_REQ	I^2C_B receive
SIU_EISR_EIF0	0x2E	SIU.SIU_EISR[EIF0]	SIU external interrupt flag 0
SIU_EISR_EIF1	0x2F	SIU.SIU_EISR[EIF1]	SIU external interrupt flag 1
I^2C_C_TX	0x30	I^2C_C.TX_REQ	I^2C_C transmit
I^2C_C_RX	0x31	I^2C_C.RX_REQ	I^2C_C receive
ADC_A	0x32		
I^2C_D_TX	0x33	I^2C_D.TX_REQ	I^2C_D transmit
I^2C_D_RX	0x34	I^2C_D.RX_REQ	I^2C_D receive
SCI_J_COMBTX	0x35	SCI_J.SCISR1[TDRE] || SCI_J.SCISR1[TC] || SCI_J.LINSTAT1[TXRDY]	SCI_J combined DMA request of the transmit data register empty, transmit complete, and LIN transmit data ready DMA requests
SCI_J_COMBRX	0x36	SCI_J.SCISR1[RDRF] || SCI_J.LINSTAT1[RXRDY]	SCI_J combined DMA request of the receive data register full and LIN receive data ready DMA requests
SCI_K_COMBTX	0x37	SCI_K.SCISR1[TDRE] || SCI_K.SCISR1[TC] || SCI_K.LINSTAT1[TXRDY]	SCI_K combined DMA request of the transmit data register empty, transmit complete, and LIN transmit data ready DMA requests
SCI_K_COMBRX	0x38	SCI_K.SCISR1[RDRF] || SCI_K.LINSTAT1[RXRDY]	SCI_K combined DMA request of the receive data register full and LIN receive data ready DMA requests
SCI_L_COMBTX	0x39	SCI_L.SCISR1[TDRE] || SCI_L.SCISR1[TC] || SCI_L.LINSTAT1[TXRDY]	SCI_L combined DMA request of the transmit data register empty, transmit complete, and LIN transmit data ready DMA requests
SCI_L_COMBRX	0x3A	SCI_L.SCISR1[RDRF] || SCI_L.LINSTAT1[RXRDY]	SCI_L combined DMA request of the receive data register full and LIN receive data ready DMA requests
SCI_M_COMBTX	0x3B	SCI_M.SCISR1[TDRE] || SCI_M.SCISR1[TC] || SCI_M.LINSTAT1[TXRDY]	SCI_M combined DMA request of the transmit data register empty, transmit complete, and LIN transmit data ready DMA requests
SCI_M_COMBRX	0x3C	SCI_M.SCISR1[RDRF] || SCI_M.LINSTAT1[RXRDY]	SCI_M combined DMA request of the receive data register full and LIN receive data ready DMA requests
Always enabled	0x3D	Always enabled	Always enabled

Table 23-4. DMA Source Configuration (continued)

DMA Request	DMA_MUX Source Input Number	DMA Source	Description
Always enabled	0x3E	Always enabled	Always enabled
Always enabled	0x3F	Always enabled	Always enabled

[1] Configuring a DMA channel to select source 0 or any of the reserved sources will disable that DMA channel.

23.4 Functional Description

The primary purpose of the DMA_MUX is to provide flexibility in the system's use of the available DMA channels. As such, configuration of the DMA_MUX is intended to be a static procedure done during execution of the system boot code. However, if the procedure outlined in Section 23.5.2, Enabling and Configuring Sources, is followed, the configuration of the DMA_MUX may be changed during the normal operation of the system.

Functionally, the DMA_MUX channels may be divided into two classes: channels 0–7, which implement the normal routing functionality and periodic triggering capability, and channels 8–31, which implement only the normal routing functionality.

23.4.1 DMA Channels 0–7

In addition to the normal routing functionality, channels 0–7 of the DMA_MUX provide a special periodic triggering capability that can be used to provide an automatic mechanism to transmit bytes, frames, or packets at fixed intervals without the need for processor intervention. The trigger is generated by the periodic interrupt timer (PIT); as such, the configuration of the periodic triggering interval is done via configuration registers in the PIT. Please refer to the periodic interrupt timer block guide for more information on this topic.

NOTE

Because of the dynamic nature of the system (i.e.,DMA channel priorities, bus arbitration, interrupt service routine lengths, etc.), the number of clock cycles between a trigger and the actual DMA transfer cannot be guaranteed.

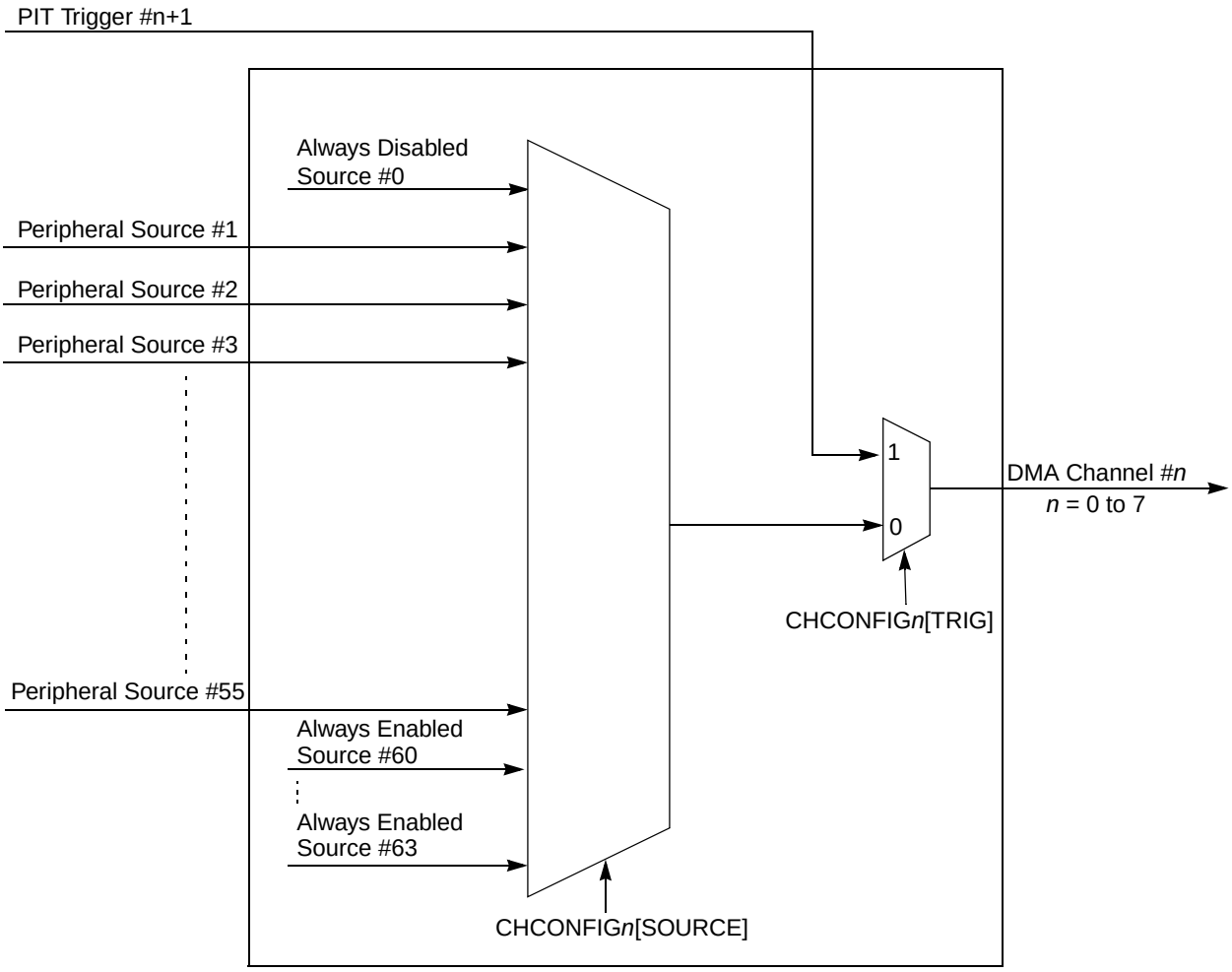


Figure 23-3. DMA_MUX Channel 0–7 Block Diagram

The DMA channel triggering capability allows the system to schedule regular DMA transfers, usually on the transmit side of certain peripherals, without the intervention of the processor. This trigger works by gating the request from the peripheral to the DMA until a trigger event has been seen. This is illustrated in Figure 23-4.

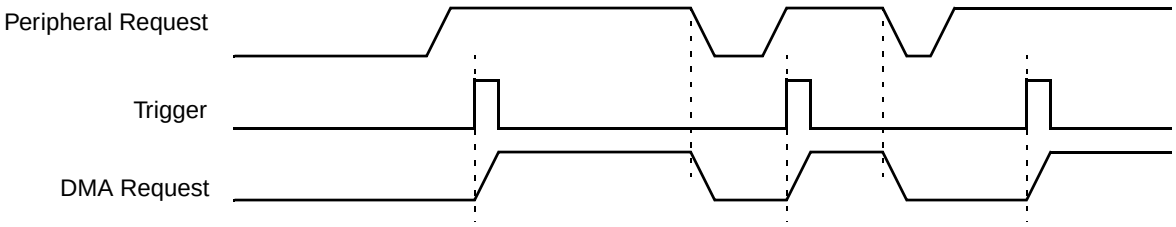


Figure 23-4. DMA_MUX Channel Triggering: Normal Operation

After the DMA request has been serviced, the peripheral negates its request, effectively resetting the gating mechanism until the peripheral re-asserts its request AND the next trigger event is seen. This means that if a trigger is seen, but the peripheral is not requesting a transfer, that triggered is ignored. This situation is illustrated in Figure 23-5.

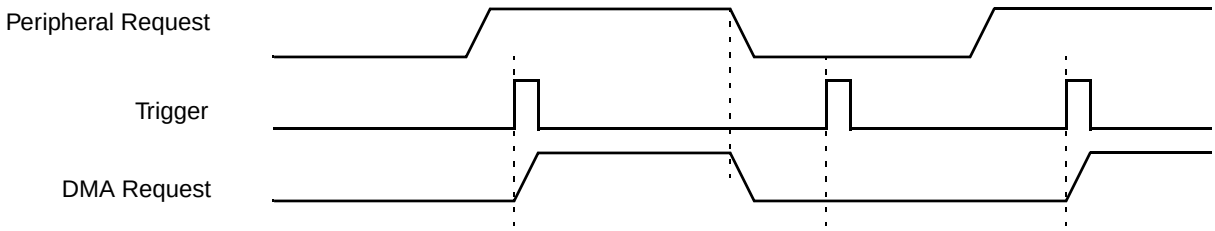


Figure 23-5. DMA_MUX Channel Triggering: Ignored Trigger

This triggering capability may be used with any peripheral that supports DMA transfers and is most useful for two types of situations:

- Periodically polling external devices on a particular bus. As an example, the transmit side of an SPI is assigned to a DMA channel with a trigger, as described above. After setup, the SPI requests DMA transfers (presumably from memory) as long as its transmit buffer is empty. By using a trigger on this channel, the SPI transfers can be automatically performed every 5 μs (as an example). On the receive side of the SPI, the SPI and DMA can be configured to transfer receive data into memory, effectively implementing a method to periodically read data from external devices and transfer the results into memory without processor intervention.
- Using the GPIO ports to drive or sample waveforms. By configuring the DMA to transfer data to one or more GPIO ports, it is possible to create complex waveforms using tabular data stored in on-chip memory. Conversely, using the DMA to periodically transfer data from one or more GPIO ports, it is possible to sample complex waveforms and store the results in tabular form in on-chip memory.

A more detailed description of the capability of each trigger (e.g., resolution, range of values, etc.) may be found in Chapter 22, Periodic Interrupt Timer (PIT).

23.4.2 DMA Channels 8–31

Channels 8–31 of the DMA_MUX provide the normal routing functionality as described in Section 23.1.3, Modes of Operation.

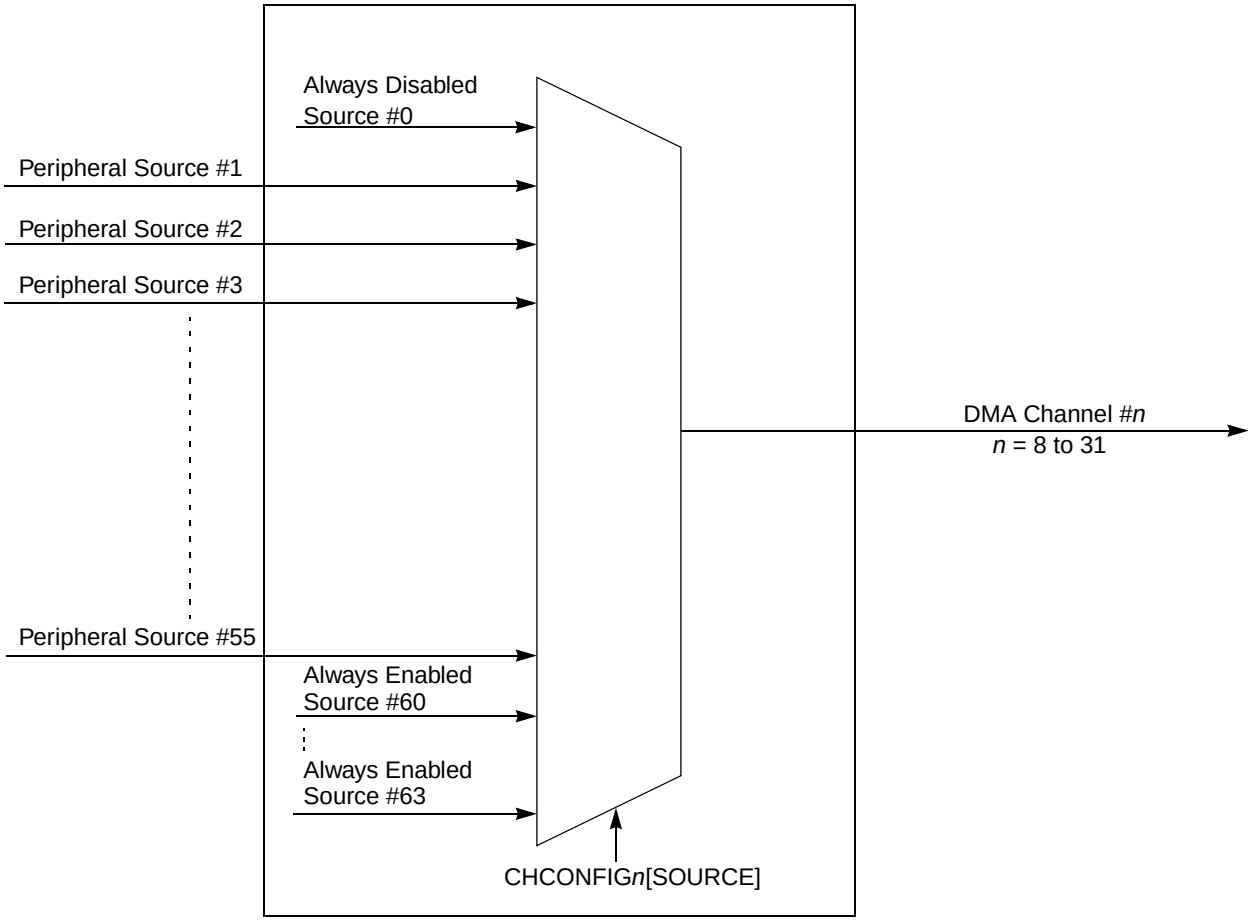


Figure 23-6. DMA_MUX Channel 8–31 Block Diagram

23.4.3 Always Enabled DMA Sources

In addition to the 55 peripherals that can be used as DMA sources, there are four additional DMA sources that are always enabled. Unlike the peripheral DMA sources, where the peripheral controls the flow of data during DMA transfers, the always enabled sources provide no such throttling of the data transfers. These sources are most useful in the following cases:

- Doing DMA transfers to/from GPIO—Moving data from/to one or more GPIO pins, either un-throttled (i.e.,as fast as possible), or periodically (using the DMA triggering capability).
- Doing DMA transfers from memory to memory—Moving data from memory to memory, typically as fast as possible, sometimes with software activation.
- Doing DMA transfers from memory to the external bus (or vice-versa)—Similar to memory to memory transfers, this is typically done as quickly as possible.
- Any DMA transfer that requires software activation—Any DMA transfer that should be explicitly started by software.

In cases where software should initiate the start of a DMA transfer, an always enabled DMA source can be used to provide maximum flexibility. When activating a DMA channel via software, subsequent

executions of the minor loop require a new start event be sent. This can either be a new software activation or a transfer request from the DMA channel mux. The options for doing this are:

- Transfer all data in a single minor loop. By configuring the DMA to transfer all of the data in a single minor loop (i.e.,major loop counter = 1), no re-activation of the channel is necessary. The disadvantage to this option is the reduced granularity in determining the load that the DMA transfer will incur on the system. For this option, the DMA channel should be disabled in the DMA channel mux.
- Use explicit software re-activation. In this option, the DMA is configured to transfer the data using both minor and major loops, but the processor is required to re-activate the channel (by writing to the DMA registers) after every minor loop. For this option, the DMA channel should be disabled in the DMA channel mux.
- Use an always enabled DMA source. In this option, the DMA is configured to transfer the data using both minor and major loops, and the DMA channel mux does the channel re-activation. For this option, the DMA channel should be enabled and pointing to an always enabled source. Note that the re-activation of the channel can be continuous (DMA triggering is disabled) or can use the DMA triggering capability. In this manner, it is possible to execute periodic transfers of packets of data from one source to another without processor intervention.

23.5 Initialization/Application Information

23.5.1 Reset

The reset state of each individual bit is shown within the register description section (Section 23.3.2, Register Descriptions). After reset, all channels are disabled and must be explicitly enabled before use.

23.5.2 Enabling and Configuring Sources

23.5.2.1 Enabling a Source with Periodic Triggering

1. Determine with which DMA channel the source will be associated. Only DMA channels 0–7 have periodic triggering capability.
2. Clear the ENBL and TRIG bits of the DMA channel.
3. Ensure that the DMA channel is properly configured in the DMA. The DMA channel may be enabled at this point.
4. In the PIT, configure the corresponding timer.
5. Select the source to be routed to the DMA channel. Write to the corresponding CHCONFIG register, ensuring that the ENBL and TRIG bits are set.

Example 23-1. Configure DSPI_B Transmit for use with DMA Channel 2, with periodic triggering capability

1. Write 0x00 to CHCONFIG2 (base address + 0x02).
2. Configure channel 2 in the DMA, including enabling the channel.
3. Configure timer 3 in the periodic interrupt timer (PIT) for the desired trigger interval.
4. Write 0xD3 to CHCONFIG2 (base address + 0x02).

The following code example illustrates steps #1 and #4 above:

```
In File registers.h:
#define DMAMUX_BASE_ADDR     0xFFFDC000 /* Base addr for PXN20 */
/* Following example assumes char is 8-bits */
volatile unsigned char *CHCONFIG0 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0000);
volatile unsigned char *CHCONFIG1 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0001);
volatile unsigned char *CHCONFIG2 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0002);
volatile unsigned char *CHCONFIG3 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0003);
volatile unsigned char *CHCONFIG4 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0004);
volatile unsigned char *CHCONFIG5 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0005);
volatile unsigned char *CHCONFIG6 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0006);
volatile unsigned char *CHCONFIG7 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0007);
volatile unsigned char *CHCONFIG8 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0008);
volatile unsigned char *CHCONFIG9 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0009);
volatile unsigned char *CHCONFIG10= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x000A);
volatile unsigned char *CHCONFIG11= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x000B);
volatile unsigned char *CHCONFIG12= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x000C);
volatile unsigned char *CHCONFIG13= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x000D);
volatile unsigned char *CHCONFIG14= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x000E);
volatile unsigned char *CHCONFIG15= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x000F);
volatile unsigned char *CHCONFIG16= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0010);
volatile unsigned char *CHCONFIG17= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0011);
volatile unsigned char *CHCONFIG18= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0012);
volatile unsigned char *CHCONFIG19= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0013);
volatile unsigned char *CHCONFIG20= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0014);
volatile unsigned char *CHCONFIG21= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0015);
volatile unsigned char *CHCONFIG22= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0016);
volatile unsigned char *CHCONFIG23= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0017);
volatile unsigned char *CHCONFIG24= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0018);
volatile unsigned char *CHCONFIG25= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0019);
volatile unsigned char *CHCONFIG26= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x001A);
volatile unsigned char *CHCONFIG27= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x001B);
volatile unsigned char *CHCONFIG28= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x001C);
volatile unsigned char *CHCONFIG29= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x001D);
volatile unsigned char *CHCONFIG30= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x001E);
volatile unsigned char *CHCONFIG31= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x001F);

In File main.c:
#include “registers.h”
         :
         :
*CHCONFIG2 = 0x00;
*CHCONFIG2 = 0xD3;
```

23.5.2.2 Enabling a Source without Periodic Triggering

1. Determine with which DMA channel the source will be associated. Only DMA channels 0–7 have periodic triggering capability.
2. Clear the ENBL and TRIG bits of the DMA channel.
3. Ensure that the DMA channel is properly configured in the DMA. The DMA channel may be enabled at this point.

4. Select the source to be routed to the DMA channel. Write to the corresponding CHCONFIG register, ensuring that the ENBL is set and the TRIG bit is cleared.

Example 23-2. Configure DSPI_B Transmit for use with DMA Channel 2, with no periodic triggering capability.

1. Write 0x00 to CHCONFIG2 (base address + 0x02).
2. Configure channel 2 in the DMA, including enabling the channel.
3. Write 0x93 to CHCONFIG2 (base address + 0x02).

The following code example illustrates steps #1 and #3 above:

```
In File registers.h:
#define DMAMUX_BASE_ADDR     0xFFFDC000/* Base addr for PXN20 */
/* Following example assumes char is 8-bits */
volatile unsigned char *CHCONFIG0 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0000);
volatile unsigned char *CHCONFIG1 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0001);
volatile unsigned char *CHCONFIG2 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0002);
volatile unsigned char *CHCONFIG3 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0003);
volatile unsigned char *CHCONFIG4 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0004);
volatile unsigned char *CHCONFIG5 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0005);
volatile unsigned char *CHCONFIG6 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0006);
volatile unsigned char *CHCONFIG7 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0007);
volatile unsigned char *CHCONFIG8 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0008);
volatile unsigned char *CHCONFIG9 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0009);
volatile unsigned char *CHCONFIG10= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x000A);
volatile unsigned char *CHCONFIG11= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x000B);
volatile unsigned char *CHCONFIG12= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x000C);
volatile unsigned char *CHCONFIG13= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x000D);
volatile unsigned char *CHCONFIG14= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x000E);
volatile unsigned char *CHCONFIG15= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x000F);
volatile unsigned char *CHCONFIG16= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0010);
volatile unsigned char *CHCONFIG17= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0011);
volatile unsigned char *CHCONFIG18= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0012);
volatile unsigned char *CHCONFIG19= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0013);
volatile unsigned char *CHCONFIG20= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0014);
volatile unsigned char *CHCONFIG21= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0015);
volatile unsigned char *CHCONFIG22= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0016);
volatile unsigned char *CHCONFIG23= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0017);
volatile unsigned char *CHCONFIG24= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0018);
volatile unsigned char *CHCONFIG25= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0019);
volatile unsigned char *CHCONFIG26= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x001A);
volatile unsigned char *CHCONFIG27= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x001B);
volatile unsigned char *CHCONFIG28= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x001C);
volatile unsigned char *CHCONFIG29= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x001D);
volatile unsigned char *CHCONFIG30= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x001E);
volatile unsigned char *CHCONFIG31= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x001F);

In File main.c:
#include “registers.h”
         :
         :
*CHCONFIG2 = 0x00;
*CHCONFIG2 = 0x93;
```

23.5.2.3 Disabling a Source

A particular DMA source may be disabled by not writing the corresponding source value into any of the CHCONFIG registers. Some module specific configuration may also be necessary. Refer to the appropriate section for more details.

23.5.2.4 Switching the Source of a DMA Channel

1. Disable the DMA channel in the DMA and re-configure the channel for the new source.
2. Clear the ENBL and TRIG bits of the DMA channel.
3. Select the source to be routed to the DMA channel. Write to the corresponding CHCONFIG register, ensuring that the ENBL and TRIG bits are set.

Example 23-3. Switch DMA Channel 8 from DSPI_A transmit to ESCI_A transmit

1. In the DMA configuration registers, disable DMA channel 8 and re-configure it to handle the DSPI_A transmits.
2. Write 0x00 to CHCONFIG8 (base address + 0x08).
3. Write 0x82 to CHCONFIG8 (base address + 0x08). In this case, setting the TRIG bit has no effect because channels 8–31 do not support the periodic triggering functionality.

The following code example illustrates steps #2 and #3 above:

```
In File registers.h:
#define DMAMUX_BASE_ADDR     0xFFFDC000/* Base addr for PXN20 */
/* Following example assumes char is 8-bits */
volatile unsigned char *CHCONFIG0 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0000);
volatile unsigned char *CHCONFIG1 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0001);
volatile unsigned char *CHCONFIG2 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0002);
volatile unsigned char *CHCONFIG3 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0003);
volatile unsigned char *CHCONFIG4 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0004);
volatile unsigned char *CHCONFIG5 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0005);
volatile unsigned char *CHCONFIG6 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0006);
volatile unsigned char *CHCONFIG7 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0007);
volatile unsigned char *CHCONFIG8 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0008);
volatile unsigned char *CHCONFIG9 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0009);
volatile unsigned char *CHCONFIG10= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x000A);
volatile unsigned char *CHCONFIG11= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x000B);
volatile unsigned char *CHCONFIG12= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x000C);
volatile unsigned char *CHCONFIG13= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x000D);
volatile unsigned char *CHCONFIG14= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x000E);
volatile unsigned char *CHCONFIG15= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x000F);
volatile unsigned char *CHCONFIG16= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0010);
volatile unsigned char *CHCONFIG17= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0011);
volatile unsigned char *CHCONFIG18= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0012);
volatile unsigned char *CHCONFIG19= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0013);
volatile unsigned char *CHCONFIG20= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0014);
volatile unsigned char *CHCONFIG21= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0015);
volatile unsigned char *CHCONFIG22= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0016);
volatile unsigned char *CHCONFIG23= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0017);
volatile unsigned char *CHCONFIG24= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0018);
volatile unsigned char *CHCONFIG25= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0019);
volatile unsigned char *CHCONFIG26= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x001A);
volatile unsigned char *CHCONFIG27= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x001B);
```

```
volatile unsigned char *CHCONFIG28= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x001C);
volatile unsigned char *CHCONFIG29= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x001D);
volatile unsigned char *CHCONFIG30= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x001E);
volatile unsigned char *CHCONFIG31= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x001F);

In File main.c:
#include “registers.h”
         :
         :
*CHCONFIG8 = 0x00;
*CHCONFIG8 = 0x82;
```

23.6 Interrupts

The DMA channel mux does not generate interrupts.

Chapter 24
Enhanced Direct Memory Access Controller (eDMA)

24.1 Introduction

The enhanced direct memory access controller (eDMA) is a second-generation platform block capable of performing complex data movements through 32 programmable channels, with minimal intervention from the host processor. The hardware microarchitecture includes a DMA engine that performs source and destination address calculations, and the actual data movement operations, along with an SRAM-based memory containing the transfer control descriptors (TCD) for the channels. This implementation minimizes the overall block size.

24.1.1 Block Diagram

Figure 24-1 shows a simplified block diagram of the eDMA.

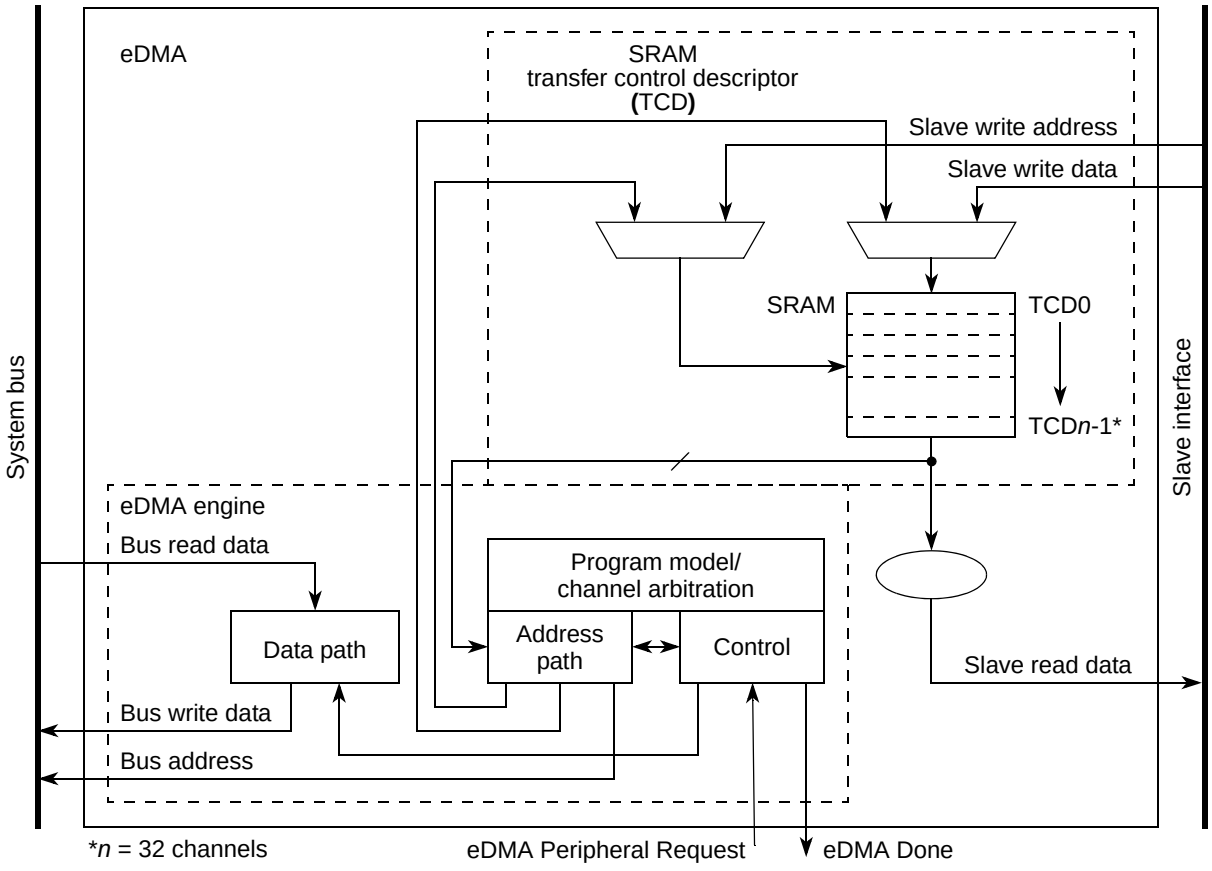


Figure 24-1. eDMA Block Diagram

24.1.2 Features

The eDMA has these major features:

- All data movement via dual-address transfers: read from source, write to destination
  - Programmable source, destination addresses, transfer size, and support for enhanced addressing modes
- Transfer control descriptor organized to support two-deep, nested transfer operations
  - An inner data transfer loop defined by a minor byte transfer count
  - An outer data transfer loop defined by a major iteration count
- Channel activation via one of three methods:
  - Explicit software initiation
  - Initiation via a channel-to-channel linking mechanism for continuous transfers
  - Peripheral-paced hardware requests (one per channel)

    All three methods require one activation per execution of the minor loop
- Support for fixed-priority and round-robin channel arbitration
- Channel completion reported via optional interrupt requests
  - One interrupt per channel, optionally asserted at completion of major iteration count
  - Error terminations are optionally enabled per channel and logically summed together to form a single error interrupt.
- Support for scatter-gather DMA processing
- Support for complex data structures
- Support to cancel transfers via software or hardware
- Any channel can be programmed to be suspended by a higher priority channel's activation, before completion of a minor loop.

NOTE

eDMA channels 16–31 are not implemented on the PXN20.

24.1.3 Modes of Operation

There are two main operating modes of eDMA: normal mode and debug mode. These modes are briefly described in this section.

24.1.3.1 Normal Mode

In normal mode, the eDMA is used to transfer data between a source and a destination. The source and destination can be a memory block or an I/O block capable of operation with the eDMA.

24.1.3.2 Debug Mode

In debug mode, the eDMA does not accept new transfer requests when its debug input signal is asserted. If the signal is asserted during transfer of a block of data described by a minor loop in the current active channel's TCD, the eDMA continues operation until completion of the minor loop.

24.2 External Signal Description

The eDMA has no external signals.

24.3 Memory Map and Registers

This section provides a detailed description of all eDMA registers.

24.3.1 Module Memory Map

The eDMA memory map is shown in Table 24-1. The address of each register is given as an offset to the eDMA base address. Registers are listed in address order, identified by complete name and mnemonic, and list the type of accesses allowed. Table 24-2 shows a graphical representation of the same memory map.

The eDMA's programming model is partitioned into two regions: the first region defines a number of registers providing control functions; however, the second region corresponds to the local transfer control descriptor memory.

Some registers are implemented as two 32-bit registers, and include H and L suffixes, signaling the high and low portions of the control function.

Table 24-1. eDMA Memory Map

Offset from EDMA_BASE (0xFFF4_4000)	Register	Access	Reset Value	Section/Page	Size
0x0000	EDMA_CR—eDMA control register	R/W	0x0000_0400	24.3.2.1/24-8	32
0x0004	EDMA_ESR—eDMA error status register	R	0x0000_0000	24.3.2.2/24-10	32
0x0008	Reserved				
0x000C	EDMA_ERQRL—eDMA enable request register (channels 31–00)	R/W	0x0000_0000	24.3.2.3/24-12	32
0x0010	Reserved				
0x0014	EDMA_EEIRL—eDMA enable error interrupt register (channels 31–00)	R/W	0x0000_0000	24.3.2.4/24-13	32
0x0018	EDMA_SERQR—eDMA set enable request register	W	0x00	24.3.2.5/24-14	8
0x0019	EDMA_CERQR—eDMA clear enable request register	W	0x00	24.3.2.6/24-15	8
0x001A	EDMA_SEEIR—eDMA set enable error interrupt register	W	0x00	24.3.2.7/24-15	8
0x001B	EDMA_CEEIR—eDMA clear enable error interrupt register	W	0x00	24.3.2.8/24-16	8
0x001C	EDMA_CIRQR—eDMA clear interrupt request register	W	0x00	24.3.2.9/24-17	8

Table 24-1. eDMA Memory Map (continued)

Offset from EDMA_BASE (0xFFF4_4000)	Register	Access	Reset Value	Section/Page	Size
0x001D	EDMA_CER—eDMA clear error register	W	0x00	24.3.2.10/24-18	8
0x001E	EDMA_SSBR—eDMA set start bit register	W	0x00	24.3.2.11/24-18	8
0x001F	EDMA_CDSBR—eDMA clear done status bit register	W	0x00	24.3.2.12/24-19	8
0x0020	Reserved				
0x0024	EDMA_IRQRL—eDMA interrupt request register (channels 31–00)	R/W	0x0000_0000	24.3.2.13/24-19	32
0x0028	Reserved				
0x002C	EDMA_ERL—eDMA error register (channels 31–00)	R/W	0x0000_0000	24.3.2.14/24-20	32
0x0030	Reserved				
0x0034	EDMA_HRSL—eDMA hardware request status register (channels 31–00)	R	0x0000_0000	24.3.2.15/24-21	32
0x0038–0x00FF	Reserved				
0x0100	EDMA_CPR0—eDMA channel 0 priority register	R/W	0x00	24.3.2.16/24-22	8
0x0101	EDMA_CPR1—eDMA channel 1 priority register	R/W	0x01	24.3.2.16/24-22	8
0x0102	EDMA_CPR2—eDMA channel 2 priority register	R/W	0x02	24.3.2.16/24-22	8
0x0103	EDMA_CPR3—eDMA channel 3 priority register	R/W	0x03	24.3.2.16/24-22	8
0x0104	EDMA_CPR4—eDMA channel 4 priority register	R/W	0x04	24.3.2.16/24-22	8
0x0105	EDMA_CPR5—eDMA channel 5 priority register	R/W	0x05	24.3.2.16/24-22	8
0x0106	EDMA_CPR6—eDMA channel 6 priority register	R/W	0x06	24.3.2.16/24-22	8
0x0107	EDMA_CPR7—eDMA channel 7 priority register	R/W	0x07	24.3.2.16/24-22	8
0x0108	EDMA_CPR8—eDMA channel 8 priority register	R/W	0x08	24.3.2.16/24-22	8
0x0109	EDMA_CPR9—eDMA channel 9 priority register	R/W	0x09	24.3.2.16/24-22	8
0x010A	EDMA_CPR10—eDMA channel 10 priority register	R/W	0x0A	24.3.2.16/24-22	8
0x010B	EDMA_CPR11—eDMA channel 11 priority register	R/W	0x0B	24.3.2.16/24-22	8
0x010C	EDMA_CPR12—eDMA channel 12 priority register	R/W	0x0C	24.3.2.16/24-22	8
0x010D	EDMA_CPR13—eDMA channel 13 priority register	R/W	0x0D	24.3.2.16/24-22	8
0x010E	EDMA_CPR14—eDMA channel 14 priority register	R/W	0x0E	24.3.2.16/24-22	8
0x010F	EDMA_CPR15—eDMA channel 15 priority register	R/W	0x0F	24.3.2.16/24-22	8
0x0110	EDMA_CPR16—eDMA channel 16 priority register	R/W	0x10	24.3.2.16/24-22	8
0x0111	EDMA_CPR17—eDMA channel 17 priority register	R/W	0x11	24.3.2.16/24-22	8
0x0112	EDMA_CPR18—eDMA channel 18 priority register	R/W	0x12	24.3.2.16/24-22	8
0x0113	EDMA_CPR19—eDMA channel 19 priority register	R/W	0x13	24.3.2.16/24-22	8

Table 24-1. eDMA Memory Map (continued)

Offset from EDMA_BASE (0xFFF4_4000)	Register	Access	Reset Value	Section/Page	Size
0x0114	EDMA_CPR20—eDMA channel 20 priority register	R/W	0x14	24.3.2.16/24-22	8
0x0115	EDMA_CPR21—eDMA channel 21 priority register	R/W	0x15	24.3.2.16/24-22	8
0x0116	EDMA_CPR22—eDMA channel 22 priority register	R/W	0x16	24.3.2.16/24-22	8
0x0117	EDMA_CPR23—eDMA channel 23 priority register	R/W	0x17	24.3.2.16/24-22	8
0x0118	EDMA_CPR24—eDMA channel 24 priority register	R/W	0x18	24.3.2.16/24-22	8
0x0119	EDMA_CPR25—eDMA channel 25 priority register	R/W	0x19	24.3.2.16/24-22	8
0x011A	EDMA_CPR26—eDMA channel 26 priority register	R/W	0x1A	24.3.2.16/24-22	8
0x011B	EDMA_CPR27—eDMA channel 27 priority register	R/W	0x1B	24.3.2.16/24-22	8
0x011C	EDMA_CPR28—eDMA channel 28 priority register	R/W	0x1C	24.3.2.16/24-22	8
0x011D	EDMA_CPR29—eDMA channel 29 priority register	R/W	0x1D	24.3.2.16/24-22	8
0x011E	EDMA_CPR30—eDMA channel 30 priority register	R/W	0x1E	24.3.2.16/24-22	8
0x011F	EDMA_CPR31—eDMA channel 31 priority register	R/W	0x1F	24.3.2.16/24-22	8
0x0120–0x0FFF	Reserved				
0x1000	TCD00—eDMA transfer control descriptor 00	R/W	—[1]	24.3.2.17/24-23	256
0x1020	TCD01—eDMA transfer control descriptor 01	R/W	—[1]	24.3.2.17/24-23	256
0x1040	TCD02—eDMA transfer control descriptor 02	R/W	—[1]	24.3.2.17/24-23	256
0x1060	TCD03—eDMA transfer control descriptor 03	R/W	—[1]	24.3.2.17/24-23	256
0x1080	TCD04—eDMA transfer control descriptor 04	R/W	—[1]	24.3.2.17/24-23	256
0x10A0	TCD05—eDMA transfer control descriptor 05	R/W	—[1]	24.3.2.17/24-23	256
0x10C0	TCD06—eDMA transfer control descriptor 06	R/W	—[1]	24.3.2.17/24-23	256
0x10E0	TCD07—eDMA transfer control descriptor 07	R/W	—[1]	24.3.2.17/24-23	256
0x1100	TCD08—eDMA transfer control descriptor 08	R/W	—[1]	24.3.2.17/24-23	256
0x1120	TCD09—eDMA transfer control descriptor 09	R/W	—[1]	24.3.2.17/24-23	256
0x1140	TCD10—eDMA transfer control descriptor 10	R/W	—[1]	24.3.2.17/24-23	256
0x1160	TCD11—eDMA transfer control descriptor 11	R/W	—[1]	24.3.2.17/24-23	256
0x1180	TCD12—eDMA transfer control descriptor 12	R/W	—[1]	24.3.2.17/24-23	256
0x11A0	TCD13—eDMA transfer control descriptor 13	R/W	—[1]	24.3.2.17/24-23	256
0x11C0	TCD14—eDMA transfer control descriptor 14	R/W	—[1]	24.3.2.17/24-23	256
0x11E0	TCD15—eDMA transfer control descriptor 15	R/W	—[1]	24.3.2.17/24-23	256
0x1200	TCD16—eDMA transfer control descriptor 16	R/W	—[1]	24.3.2.17/24-23	256
0x1220	TCD17—eDMA transfer control descriptor 17	R/W	—[1]	24.3.2.17/24-23	256

Table 24-1. eDMA Memory Map (continued)

Offset from EDMA_BASE (0xFFF4_4000)	Register	Access	Reset Value	Section/Page	Size
0x1240	TCD18—eDMA transfer control descriptor 18	R/W	—[1]	24.3.2.17/24-23	256
0x1260	TCD19—eDMA transfer control descriptor 19	R/W	—[1]	24.3.2.17/24-23	256
0x1280	TCD20—eDMA transfer control descriptor 20	R/W	—[1]	24.3.2.17/24-23	256
0x12A0	TCD21—eDMA transfer control descriptor 21	R/W	—[1]	24.3.2.17/24-23	256
0x12C0	TCD22—eDMA transfer control descriptor 22	R/W	—[1]	24.3.2.17/24-23	256
0x12E0	TCD23—eDMA transfer control descriptor 23	R/W	—[1]	24.3.2.17/24-23	256
0x1300	TCD24—eDMA transfer control descriptor 24	R/W	—[1]	24.3.2.17/24-23	256
0x1320	TCD25—eDMA transfer control descriptor 25	R/W	—[1]	24.3.2.17/24-23	256
0x1340	TCD26—eDMA transfer control descriptor 26	R/W	—[1]	24.3.2.17/24-23	256
0x1360	TCD27—eDMA transfer control descriptor 27	R/W	—[1]	24.3.2.17/24-23	256
0x1380	TCD28—eDMA transfer control descriptor 28	R/W	—[1]	24.3.2.17/24-23	256
0x13A0	TCD29—eDMA transfer control descriptor 29	R/W	—[1]	24.3.2.17/24-23	256
0x13C0	TCD30—eDMA transfer control descriptor 30	R/W	—[1]	24.3.2.17/24-23	256
0x13E0	TCD31—eDMA transfer control descriptor 31	R/W	—[1]	24.3.2.17/24-23	256
0x1400–0x17FF	Reserved				

[1] See specific register description.

Table 24-2. eDMA 32-bit Memory Map—Graphical View

Address	Register			
0xFFF4_4000	eDMA Control Register (EDMA_CR)			
0xFFF4_4004	eDMA Error Status (EDMA_ESR)			
0xFFF4_4008	Reserved			
0xFFF4_400C	eDMA Enable Request (EDMA_ERQRL, channels 31–16)		eDMA Enable Request (EDMA_ERQRL, channels 15–00)	
0xFFF4_4010	Reserved			
0xFFF4_4014	eDMA Enable Error Interrupt Low (EDMA_EEIRL, channels 31–16)		eDMA Enable Error Interrupt Low (EDMA_EEIRL, Channels 15–00)	
0xFFF4_4018	eDMA Set Enable Request (EDMA_SERQR)	eDMA Clear Enable Request (EDMA_CERQR)	eDMA Set Enable Error Interrupt (EDMA_SEEIR)	eDMA Clear Enable Error Interrupt (EDMA_CEEIR)
0xFFF4_401C	eDMA Clear Interrupt Request (EDMA_CIRQR)	eDMA Clear Error (EDMA_CER)	eDMA Set Start Bit, Activate Channel (EDMA_SSBR)	eDMA Clear Done Status Bit (EDMA_CDSBR)
0xFFF4_4020	Reserved			

Table 24-2. eDMA 32-bit Memory Map—Graphical View (continued)

Address	Register			
0xFFF4_4024	eDMA Interrupt Request (EDMA_IRQRL, channels 31–16)		eDMA Interrupt Request (EDMA_IRQRL, Channels 15–00)	
0xFFF4_4028	Reserved			
0xFFF4_402C	eDMA Error (EDMA_ERL, channels 31–16)		eDMA Error (EDMA_ERL, Channels 15–00)	
0xFFF4_4030	Reserved			
0xFFF4_4034	eDMA Hardware Request Status (EDMA_HRSL, channels 31–16)		eDMA Hardware Request Status (EDMA_HRSL, Channels 15–00)	
0xFFF4_4038 – 0xFFF4_40FC	Reserved			
0xFFF4_4100	eDMA Channel 0 Priority (EDMA_CPR0)	eDMA Channel 1 Priority (EDMA_CPR1)	eDMA Channel 2 Priority (EDMA_CPR2)	eDMA Channel 3 Priority (EDMA_CPR3)
0xFFF4_4104	eDMA Channel 4 Priority (EDMA_CPR4)	eDMA Channel 5 Priority (EDMA_CPR5)	eDMA Channel 6 Priority (EDMA_CPR6)	eDMA Channel 7 Priority EDMA_CPR7)
0xFFF4_4108	eDMA Channel 8 Priority (EDMA_CPR8)	eDMA Channel 9 Priority (EDMA_CPR9)	eDMA Channel 10 Priority (EDMA_CPR10)	eDMA Channel 11 Priority (EDMA_CPR11)
0xFFF4_410C	eDMA Channel 12 Priority (EDMA_CPR12)	eDMA Channel 13 Priority (EDMA_CPR13)	eDMA Channel 14 Priority (EDMA_CPR14)	eDMA Channel 15 Priority (EDMA_CPR15)
0xFFF4_4110	eDMA Channel 16 Priority (EDMA_CPR16)	eDMA Channel 17 Priority (EDMA_CPR17)	eDMA Channel 18 Priority (EDMA_CPR18)	eDMA Channel 19 Priority (EDMA_CPR19)
0xFFF4_4114	eDMA Channel 20 Priority (EDMA_CPR16)	eDMA Channel 21 Priority (EDMA_CPR17)	eDMA Channel 22 Priority (EDMA_CPR18)	eDMA Channel 23 Priority (EDMA_CPR19)
0xFFF4_4118	eDMA Channel 24 Priority (EDMA_CPR16)	eDMA Channel 25 Priority (EDMA_CPR17)	eDMA Channel 26 Priority (EDMA_CPR18)	eDMA Channel 27 Priority (EDMA_CPR19)
0xFFF4_411C	eDMA Channel 28 Priority (EDMA_CPR16)	eDMA Channel 29 Priority (EDMA_CPR17)	eDMA Channel 30 Priority (EDMA_CPR18)	eDMA Channel 31 Priority (EDMA_CPR19)
0xFFF4_5000 – 0xFFF4_51FC	TCD00–TCD15			
0xFFF4_5200 – 0xFFF4_53FC	TCD16–TCD31			
0xFFF4_5400	Reserved			

24.3.2 Register Descriptions

This section lists the eDMA registers in address order and describes the registers and their bit fields.

Reading reserved bits in a register returns the value of zero. Writes to reserved bits in a register are ignored. Reading or writing to a reserved memory location generates a bus error.

Many of the control registers have a bit width that matches the number of channels implemented in the module, or 32 bits in size.

24.3.2.1 eDMA Control Register (EDMA_CR)

The 32-bit EDMA_CR defines the basic operating configuration of the eDMA.

Arbitration among the channels can be configured to use a fixed priority or a round robin. In fixed-priority arbitration, the highest priority channel requesting service is selected to execute. The priorities are assigned by the channel priority registers. See Section 24.3.2.16, eDMA Channel n Priority Registers (EDMA_CPRn). In round-robin arbitration mode, the channel priorities are ignored and the channels within each group are cycled through, from channel 31 down to channel 0,without regard to priority.

Minor loop offsets are address offset values added to the final source address (SADDR) or destination address (DADDR) upon minor loop completion. When minor loop offsets are enabled, the minor loop offset (MLOFF) is added to the final source address (SADDR) or to the final destination address (DADDR) or to both addresses prior to the addresses being written back into the TCD. If the major loop is complete, the minor loop offset is ignored and the major loop address offsets (SLAST and DLAST_SGA) are used to compute the next TCR.SADDR and TCR.DADDR values.

When minor loop mapping is enabled (EDMA_CR[EMLM] = 1), TCD*n* word2 is redefined. A portion of TCD*n* word2 is used to specify multiple fields: a source enable bit (SMLOE) to specify that the minor loop offset should be applied to the source address (SADDR) upon minor loop completion, a destination enable bit (DMLOE) to specify the minor loop offset should be applied to the destination address (DADDR) upon minor loop completion, and the sign extended minor loop offset value (MLOFF). The same offset value (MLOFF) is used for both source and destination minor loop offsets.

When either of the minor loop offsets is enabled (SMLOE is set or DMLOE is set), the NBYTES field is reduced to 8 bits. When both minor loop offsets are disabled (SMLOE is cleared and DMLOE is cleared), the NBYTES field becomes a 30-bit vector.

When minor loop mapping is disabled (EDMA_CR[EMLM] = 0), all 32 bits of TCD*n* word2 are assigned to the NBYTES field. See Section 24.3.2.17, Transfer Control Descriptor (TCD), for more details.

Offset: EDMA_BASE + 0x0000 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	CXFR	ECX
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	GRP1 PRI	0	GRP0 PRI	EMLM	CLM	HALT	HOE	ERGA	ERCA	EDBG	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0

Figure 24-2. eDMA Control Register (EDMA_CR)

Table 24-3. EDMA_CR Field Descriptions

Field	Description
CXFR	Cancel Transfer. 0 Normal operation. 1 Cancel the remaining data transfer. Stop the executing channel and force the minor loop to be finished. The cancel takes effect after the last write of the current read/write sequence. The CXFR bit clears itself after the cancel has been honored. This cancel retires the channel normally as if the minor loop was completed.
ECX	Error cancel transfer. 0 Normal operation. 1 Cancel the remaining data transfer in the same fashion as the CXFR cancel transfer. Stop the executing channel and force the minor loop to be finished. The cancel takes effect after the last write of the current read/write sequence. The ECX bit clears itself after the cancel has been honored. In addition to cancelling the transfer, the ECX treats the cancel as an error condition; thus updating the EDMA_ESR register and generating an optional error interrupt. See Section 24.3.2.2, eDMA Error Status Register (EDMA_ESR).
GRP1PRI	Channel group 1 priority. Group 1 priority level when fixed priority group arbitration is enabled.
GRP0PRI	Channel group 0 priority. Group 0 priority level when fixed priority group arbitration is enabled.
EMLM	Enable minor loop mapping. 0 Minor loop mapping disabled. TCD Word 2 is defined as a 32-bit nbytes field. 1 Minor loop mapping enabled. When set, TCD*n* Word 2 is redefined to include individual enable fields, an offset field and the NBYTES field. The individual enable fields allow the minor loop offset to be applied to the source address, the destination address, or both. The NBYTES field is reduced when either offset is enabled.
CLM	Continuous link mode. 0 A minor loop channel link made to itself goes through channel arbitration before being activated again. 1 A minor loop channel link made to itself does not go through channel arbitration before being activated again. Upon minor loop completion, the channel is active again if that channel has a minor loop channel link enabled and the link channel is itself. This effectively applies the minor loop offsets and restarts the next minor loop.
HALT	Halt DMA operations. 0 Normal operation. 1 Stall the start of any new channels. Executing channels are allowed to complete. Channel execution resumes when the HALT bit is cleared.
HOE	Halt on error. 0 Normal operation. 1 Any error causes the HALT bit to be set. Subsequently, all service requests are ignored until the HALT bit is cleared.
ERGA	Enable round-robin group arbitration. 0 Fixed-priority arbitration is used for selection among the groups. 1 Round-robin arbitration is used for selection among the groups.
ERCA	Enable Round-Robin Channel Arbitration. 0 Fixed-priority arbitration is used for channel selection within each group. 1 Round-robin arbitration is used for channel selection within each group.
EDBG	Enable Debug. 0 The assertion of the system debug control input is ignored. 1 The assertion of the system debug control input causes the eDMA to stall the start of a new channel. Executing channels are allowed to complete. Channel execution resumes when either the system debug control input is negated or the EDBG bit is cleared.

24.3.2.2 eDMA Error Status Register (EDMA_ESR)

The EDMA_ESR provides information about the last recorded channel error. Channel errors can be caused by a configuration error (an illegal setting in the transfer control descriptor or an illegal priority register setting in fixed-arbitration mode) or an error termination to a bus master read or write cycle.

A configuration error is caused when the starting source or destination address, source or destination offsets, minor loop byte count, and the transfer size represent an inconsistent state. The addresses and offsets must be aligned on 0-modulo-transfer_size boundaries, and the minor loop byte count must be a multiple of the source and destination transfer sizes. All source reads and destination writes must be configured to the natural boundary of the programmed transfer size respectively.

In fixed-arbitration mode, a configuration error is generated when any two channel priority levels are equal and any channel is activated. The ERRCHN field is undefined for this type of error. All channel priority levels must be unique before any service requests are made.

If a scatter-gather operation is enabled on channel completion, a configuration error is reported if the scatter-gather address (DLAST_SGA) is not aligned on a 32-byte boundary. If minor loop channel linking is enabled on channel completion, a configuration error is reported when the link is attempted if the TCD.CITER.E_LINK bit is not equal to the TCD.BITER.E_LINK bit. All configuration error conditions except scatter-gather and minor loop link error are reported as the channel is activated and assert an error interrupt request if enabled. When properly enabled, a scatter-gather configuration error is reported when the scatter-gather operation begins at major loop completion. A minor loop channel link configuration error is reported when the link operation is serviced at minor loop completion.

If a system bus read or write is terminated with an error, the data transfer is immediately stopped and the appropriate bus error flag is set. In this case, the state of the channel’s transfer control descriptor is updated by the DMA engine with the current source address, destination address, and minor loop byte count at the point of the fault. If a bus error occurs on the last read prior to beginning the write sequence, the write is executed using the data captured during the bus error. If a bus error occurs on the last write prior to switching to the next read sequence, the read sequence is executed before the channel is terminated due to the destination bus error.

A transfer may be cancelled by software via the EDMA_CR[CX] bit. When a cancel transfer request is recognized, the eDMA engine stops processing the channel. The current read-write sequence is allowed to finish. If the cancel occurs on the last read-write sequence of a major or minor loop, the cancel request is discarded and the channel retires normally.

The error cancel transfer is the same as a cancel transfer except the DMAES register is updated with the cancelled channel number and error cancel bit is set. The TCD of a cancelled channel has the source address and destination address of the last transfer saved in the TCD. It is the responsibility of the user to initialize the TCD again should the channel need to be restarted because the aforementioned fields have been modified by the eDMA engine and no longer represent the original parameters. When a transfer is cancelled via the error cancel transfer mechanism (setting the EDMA_CR[ECX]), the channel number is loaded into the EDMA_ESR[ERRCHN] field and the EDMA_ESR[ECX] and EDMA_ESR[VLD] bits are set. In addition, an error interrupt may be generated if enabled. Refer to Section 24.3.2.14, eDMA Error Register (EDMA_ERL).

The occurrence of any type of error causes the DMA engine to stop the active channel and the appropriate channel bit in the eDMA error register to be asserted. At the same time, the details of the error condition are loaded into the EDMA_ESR. The major loop complete indicators, setting the transfer control descriptor DONE flag and the possible assertion of an interrupt request, are not affected when an error is detected. After the error status has been updated, the DMA engine continues to operate by servicing the next appropriate channel. A channel that experiences an error condition is not automatically disabled. If a channel is terminated by an error and then issues another service request before the error is fixed, that channel will execute and terminate with the same error condition.

Offset: EDMA_BASE + 0x0004 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	VLD	0	0	0	0	0	0	0	0	0	0	0	0	0	0	ECX
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	GPE	CPE	ERRCHN						SAE	SOE	DAE	DOE	NCE	SGE	SBE	DBE
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 24-3. eDMA Error Status Register (EDMA_ESR)

Table 24-4. EDMA_ESR Field Descriptions

Field	Description
VLD	Valid Bit. Logical OR of all EDMA_ERL status bits. 0 No EDMA_ER bits are set. 1 At least one EDMA_ER bit is set indicating a valid error exists that has not been cleared.
ECX	Transfer canceled. 0 No canceled transfers. 1 The last recorded entry was a canceled transfer via the error cancel transfer input.
GPE	Group-priority error. 0 No group-priority error. 1 The last recorded error was a configuration error among the group priorities indicating not all group priorities are unique.
CPE	Channel-Priority Error. 0 No channel-priority error. 1 The last recorded error was a configuration error in the channel priorities within a group, indicating not all channel priorities within a group are unique.
ERRCHN	Error Channel Number or Canceled Channel Number. Channel number of the last recorded error (excluding GPE and CPE errors) or last recorded transfer that was error cancelled. **Note:** Do not rely on the number in the ERRCHN field group for channel-priority errors. Group- and Channel-priority errors must be resolved by inspection. The application code must interrogate the priority registers to find groups or channels with duplicate priority level.

Table 24-4. EDMA_ESR Field Descriptions (continued)

Field	Description
SAE	Source Address Error. 0 No source address configuration error. 1 The last recorded error was a configuration error detected in the TCD.SADDR field, indicating TCD.SADDR is inconsistent with TCD.SSIZE.
SOE	Source Offset Error. 0 No source offset configuration error. 1 The last recorded error was a configuration error detected in the TCD.SOFF field, indicating TCD.SOFF is inconsistent with TCD.SSIZE.
DAE	Destination Address Error. 0 No destination address configuration error. 1 The last recorded error was a configuration error detected in the TCD.DADDR field, indicating TCD.DADDR is inconsistent with TCD.DSIZE.
DOE	Destination Offset Error. 0 No destination offset configuration error. 1 The last recorded error was a configuration error detected in the TCD.DOFF field, indicating TCD.DOFF is inconsistent with TCD.DSIZE.
NCE	NBYTES/CITER Configuration Error. 0 No NBYTES/CITER configuration error. 1 The last recorded error was a configuration error detected in the TCD.NBYTES or TCD.CITER fields, indicating the following conditions exist: • TCD.NBYTES is not a multiple of TCD.SSIZE and TCD.DSIZE, or • TCD.CITER is equal to zero, or • TCD.CITER.E_LINK is not equal to TCD.BITER.E_LINK.
SGE	Scatter-Gather Configuration Error. 0 No scatter-gather configuration error. 1 The last recorded error was a configuration error detected in the TCD.DLAST_SGA field, indicating TCD.DLAST_SGA is not on a 32-byte boundary. This field is checked at the beginning of a scatter-gather operation after major loop completion if TCD.E_SG is enabled.
SBE	Source Bus Error. 0 No source bus error. 1 The last recorded error was a bus error on a source read.
DBE	Destination Bus Error. 0 No destination bus error. 1 The last recorded error was a bus error on a destination write.

24.3.2.3 eDMA Enable Request Register (EDMA_ERQRL)

The EDMA_ERQRL provides a bit map for the 32 channels to enable the request signal for each channel. EDMA_ERQRL maps to channels 31–0.

The state of any given channel enable is directly affected by writes to this register; the state is also affected by writes to the EDMA_SERQR and EDMA_CERQR. The EDMA_CERQR and EDMA_SERQR are provided so that the request enable for a single channel can be modified without performing a read-modify-write sequence to the EDMA_ERQRL.

Both the eDMA request input signal and this enable request flag must be asserted before a channel's hardware service request is accepted. The state of the eDMA enable request flag does not effect a channel service request made through software or a linked channel request.

Offset: EDMA_BASE + 0x000C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	ERQ 31	ERQ 30	ERQ 29	ERQ 28	ERQ 27	ERQ 26	ERQ 25	ERQ 24	ERQ 23	ERQ 22	ERQ 21	ERQ 20	ERQ 19	ERQ 18	ERQ 17	ERQ 16
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	19	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	ERQ 15	ERQ 14	ERQ 13	ERQ 12	ERQ 11	ERQ 10	ERQ 09	ERQ 08	ERQ 07	ERQ 06	ERQ 05	ERQ 04	ERQ 03	ERQ 02	ERQ 01	ERQ 00
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 24-4. eDMA Enable Request Register (EDMA_ERQRL)

Table 24-5. EDMA_ERQRL Field Descriptions

Field	Description
ERQ*n*	Enable eDMA Hardware Service Request *n.* 0 The eDMA request signal for channel *n* is disabled. 1 The eDMA request signal for channel *n* is enabled.

As a given channel completes processing its major iteration count, there is a flag in the transfer control descriptor that may affect the ending state of the EDMA_ERQR bit for that channel. If the TCD.D_REQ bit is set, then the corresponding EDMA_ERQR bit is cleared after the major loop is complete, disabling the eDMA hardware request. Otherwise if the D_REQ bit is cleared, the state of the EDMA_ERQR bit is unaffected.

24.3.2.4 eDMA Enable Error Interrupt Register (EDMA_EEIRL)

The EDMA_EEIRL provides a bit map for the 32 channels to enable the error interrupt signal for each channel. EDMA_EEIRL maps to channels 31–0.

The state of any given channel's error interrupt enable is directly affected by writes to these registers; it is also affected by writes to the EDMA_SEEIR and EDMA_CEEIR. The EDMA_SEEIR and EDMA_CEEIR are provided so that the error interrupt enable for a single channel can be modified without the performing a read-modify-write sequence to the EDMA_EEIRL.

Both the eDMA error indicator and this error interrupt enable flag must be asserted before an error interrupt request for a given channel is asserted.

Offset: EDMA_BASE + 0x0014 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	EEI31	EEI30	EEI29	EEI28	EEI27	EEI26	EEI25	EEI24	EEI23	EEI22	EEI21	EEI20	EEI19	EEI18	EEI17	EEI16
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	19	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	EEI15	EEI14	EEI13	EEI12	EEI11	EEI10	EEI09	EEI08	EEI07	EEI06	EEI05	EEI04	EEI03	EEI02	EEI01	EEI00
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 24-5. eDMA Enable Error Interrupt Register (EDMA_EEIRL)

Table 24-6. EDMA_EEIRL Field Descriptions

Field	Description
EEI*n*	Enable Error Interrupt *n.* 0 The error signal for channel *n* does not generate an error interrupt. 1 The assertion of the error signal for channel *n* generate an error interrupt request.

24.3.2.5 eDMA Set Enable Request Register (EDMA_SERQR)

The EDMA_SERQR provides a simple memory-mapped mechanism to set a given bit in the EDMA_ERQRL to enable the eDMA request for a given channel. The data value on a register write causes the corresponding bit in the EDMA_ERQRL to be set. Setting bit 1 (SERQ[0]) provides a global set function, forcing the entire contents of EDMA_ERQRL to be asserted. Reads of this register return all zeroes.

If bit 0 is set, the SERQ command is ignored. This allows multiple byte registers to be written as a 32-bit word. Reads of this register return all zeroes.

Offset: EDMA_BASE + 0x0018 Access: User write-only

	0	1	2	3	4	5	6	7
R								
W	NOP	SERQ[0:6]						
Reset	0	0	0	0	0	0	0	0

Figure 24-6. eDMA Set Enable Request Register (EDMA_SERQR)

Table 24-7. EDMA_SERQR Field Descriptions

Field	Descriptions
NOP	No operation. 0 Normal operation. 1 No operation, ignore bits 1–7.
SERQ[0:6]	Set Enable Request. 0–31 Set corresponding bit in EDMA_ERQRL. 32–63 Reserved. 64–127 Set all bits in EDMA_ERQRL. **Note:** Bits 2 and 3(SERQR[1:2]) are not used.

24.3.2.6 eDMA Clear Enable Request Register (EDMA_CERQR)

The EDMA_CERQR provides a simple memory-mapped mechanism to clear a given bit in the EDMA_ERQRL to disable the eDMA request for a given channel. The data value on a register write causes the corresponding bit in the EDMA_ERQRL to be cleared. Setting bit 1 (CERQ[0]) provides a global clear function, forcing the entire contents of the EDMA_ERQRL to be zeroed, disabling all eDMA request inputs. Reads of this register return all zeroes.

If bit 0 is set, the CERQ command is ignored. This allows multiple byte registers to be written as a 32-bit word. Reads of this register return all zeroes.

Offset: EDMA_BASE + 0x0019 Access: User write-only

	0	1	2	3	4	5	6	7
R								
W	NOP	CERQ[0:6]						
Reset	0	0	0	0	0	0	0	0

Figure 24-7. eDMA Clear Enable Request Register (EDMA_CERQR)

Table 24-8. EDMA_CERQR Field Descriptions

Field	Description
NOP	No operation. 0 Normal operation. 1 No operation, ignore bits 1–7.
CERQ[0:6]	Clear Enable Request. 0–31 Clear corresponding bit in EDMA_ERQRL. 32–63 Reserved. 64–127 Clear all bits in EDMA_ERQRL. **Note:** Bits 2 and 3 (CERQR[1:2]) are not used.

24.3.2.7 eDMA Set Enable Error Interrupt Register (EDMA_SEEIR)

The EDMA_SEEIR provides a memory-mapped mechanism to set a given bit in the EDMA_EEIRL to enable the error interrupt for a given channel. The data value on a register write causes the corresponding

bit in the EDMA_EEIRL to be set. Setting bit 1 (SEEI[0]) provides a global set function, forcing the entire contents of EDMA_EEIRL to be asserted. Reads of this register return all zeroes.

If bit 0 is set, the SEEI command is ignored. This allows multiple byte registers to be written as a 32-bit word. Reads of this register return all zeroes.

Offset: EDMA_BASE + 0x001A Access: User write-only

	0	1	2	3	4	5	6	7
R								
W	NOP	SEEI[0:6]						
Reset	0	0	0	0	0	0	0	0

Figure 24-8. eDMA Set Enable Error Interrupt Register (EDMA_SEEIR)

Table 24-9. EDMA_SEEIR Field Descriptions

Field	Description
NOP	No operation. 0 Normal operation. 1 No operation, ignore bits 1–7.
SEEI[0:6]	Set Enable Error Interrupt. 0–31 Set corresponding bit in EDMA_EIRRL. 32–63 Reserved. 64–127 Set all bits in EDMA_EEIRL. **Note:** Bits 2 and 3 (SEEIR[1:2]) are not used.

24.3.2.8 eDMA Clear Enable Error Interrupt Register (EDMA_CEEIR)

The EDMA_CEEIR provides a memory-mapped mechanism to clear a given bit in the EDMA_EEIRL to disable the error interrupt for a given channel. The data value on a register write causes the corresponding bit in the EDMA_EEIRL to be cleared. Setting bit 1 (CEEI[0]) provides a global clear function, forcing the entire contents of the EDMA_EEIRL to be zeroed, disabling error interrupts for all channels. Reads of this register return all zeroes.

If bit 0 is set, the CEEI command is ignored. This allows multiple byte registers to be written as a 32-bit word. Reads of this register return all zeroes.

Offset: EDMA_BASE + 0x001B Access: User write-only

	0	1	2	3	4	5	6	7
R								
W	NOP	CEEI[0:6]						
Reset	0	0	0	0	0	0	0	0

Figure 24-9. eDMA Clear Enable Error Interrupt Register (EDMA_CEEIR)

Table 24-10. EDMA_CEEIR Field Descriptions

Field	Description
NOP	No operation. 0 Normal operation. 1 No operation, ignore bits 1–7.
CEEI[0:6]	Clear Enable Error Interrupt. 0–31 Clear corresponding bit in EDMA_EEIRL. 32–63 Reserved. 64–127 Clear all bits in EDMA_EEIRL. **Note:** Bits 2 and 3 (CEEIR[1:2]) are not used.

24.3.2.9 eDMA Clear Interrupt Request Register (EDMA_CIRQR)

The EDMA_CIRQR provides a memory-mapped mechanism to clear a given bit in the EDMA_IRQRL to disable the interrupt request for a given channel. The given value on a register write causes the corresponding bit in the EDMA_IRQRL to be cleared. Setting bit 1 (CINT[0]) provides a global clear function, forcing the entire contents of the EDMA_IRQRL to be zeroed, disabling all eDMA interrupt requests. Reads of this register return all zeroes.

If bit 0 is set, the CINT command is ignored. This allows multiple byte registers to be written as a 32-bit word. Reads of this register return all zeroes.

Offset: EDMA_BASE + 0X001C Access: User write-only

	0	1	2	3	4	5	6	7
R								
W	NOP	CINT[0:6]						
Reset	0	0	0	0	0	0	0	0

Figure 24-10. eDMA Clear Interrupt Request (EDMA_CIRQR)

Table 24-11. EDMA_CIRQR Field Descriptions

Field	Description
NOP	No operation. 0 Normal operation. 1 No operation, ignore bits 1-7.
CINT[0:6]	Clear Interrupt Request. 0–31 Clear corresponding bit in EDMA_IRQRL. 32–63 Reserved. 64–127 Clear all bits in EDMA_IRQRL. **Note:** Bits 2 and 3(CIRQR[1:2]) are not used.

24.3.2.10 eDMA Clear Error Register (EDMA_CER)

The EDMA_CER provides a memory-mapped mechanism to clear a given bit in the EDMA_ERL to disable the error condition flag for a given channel. The given value on a register write causes the corresponding bit in the EDMA_ERL to be cleared. Setting bit 1 (CERR[0]) provides a global clear function, forcing the entire contents of the EDMA_ERL to be zeroed, clearing all channel error indicators. Reads of this register return all zeroes.

If bit 0 is set, the CERR command is ignored. This allows multiple byte registers to be written as a 32-bit word. Reads of this register return all zeroes.

Offset: EDMA_BASE + 0x001D Access: User write-only

	0	1	2	3	4	5	6	7
R								
W	NOP	CERR[0:6]						
Reset	0	0	0	0	0	0	0	0

Figure 24-11. eDMA Clear Error Register (EDMA_CER)

Table 24-12. EDMA_CER Field Descriptions

Field	Description
NOP	No operation. 0 Normal operation. 1 No operation, ignore bits 1–7.
CERR[0:6]	Clear Error Indicator. 0–31 Clear corresponding bit in EDMA_ERL. 32–63 Reserved. 64–127 Clear all bits in EDMA_ERL. **Note:** Bits 2 and 3 (CER[1:2]) are not used.

24.3.2.11 eDMA Set START Bit Register (EDMA_SSBR)

The EDMA_SSBR provides a memory-mapped mechanism to set the START bit in the TCD of the given channel. The data value on a register write causes the START bit in the corresponding transfer control descriptor to be set. Setting bit 1 (SSB[0]) provides a global set function, forcing all START bits to be set. Reads of this register return all zeroes.

If bit 0 is set, the SSB command is ignored. This allows multiple byte registers to be written as a 32-bit word. Reads of this register return all zeroes.

Offset: EDMA_BASE + 0x001E Access: User write-only

	0	1	2	3	4	5	6	7
R								
W	NOP	SSB[0:6]						
Reset	0	0	0	0	0	0	0	0

Figure 24-12. eDMA Set START Bit Register (EDMA_SSBR)

Table 24-13. EDMA_SSBR Field Descriptions

Field	Description
NOP	No operation. 0 Normal operation. 1 No operation, ignore bits 1–7.
SSB[0:6]	Set START Bit (channel service request). 0–31 Set the corresponding channel's TCD START bit. 32–63 Reserved. 64–127 Set all TCD START bits. **Note:** Bits 2 and 3 (SSBR[1:2]) are not used.

24.3.2.12 eDMA Clear DONE Status Bit Register (EDMA_CDSBR)

The EDMA_CDSBR provides a memory-mapped mechanism to clear the DONE bit in the TCD of the given channel. The data value on a register write causes the DONE bit in the corresponding transfer control descriptor to be cleared. Setting bit 1 (CDSB[0]) provides a global clear function, forcing all DONE bits to be cleared.

If bit 0 is set, the CDSB command is ignored. This allows multiple byte registers to be written as a 32-bit word. Reads of this register return all zeroes.

Offset: EDMA_BASE + 0x001F Access: User write-only

	0	1	2	3	4	5	6	7
R								
W	NOP	CDSB[0:6]						
Reset	0	0	0	0	0	0	0	0

Figure 24-13. eDMA Clear DONE Status Bit Register (EDMA_CDSBR)

Table 24-14. EDMA_CDSBR Field Descriptions

Field	Description
NOP	No operation. 0 Normal operation. 1 No operation, ignore bits 1–7.
CDSB[0:6]	Clear DONE Status Bit. 0–31 Clear the corresponding channel's DONE bit. 32–63 Reserved. 64–127 Clear all TCD DONE bits. **Note:** Bits 2 and 3 (CDSBR[1:2]) are not used.

24.3.2.13 eDMA Interrupt Request Register (EDMA_IRQRL)

The EDMA_IRQRL provides a bit map for the 32 channels signaling the presence of an interrupt request for each channel. EDMA_IRQRL maps to channels 31–0.

The DMA engine signals the occurrence of a programmed interrupt on the completion of a data transfer as defined in the transfer control descriptor by setting the appropriate bit in this register. The outputs of this register are directly routed to the interrupt controller (INTC). During the execution of the interrupt service routine associated with any given channel, software must clear the appropriate bit, negating the interrupt request. Typically, a write to the EDMA_CIRQR in the interrupt service routine is used for this purpose.

The state of any given channel's interrupt request is directly affected by writes to this register; it is also affected by writes to the EDMA_CIRQR. On writes to the EDMA_IRQRL, a 1 in any bit position clears the corresponding channel's interrupt request. A 0 in any bit position has no effect on the corresponding channel's current interrupt status. The EDMA_CIRQR is provided so the interrupt request for a single channel can be cleared without performing a read-modify-write sequence to the EDMA_IRQRL.

Offset: EDMA_BASE + 0x0024 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	INT 31	INT 30	INT 29	INT 28	INT 27	INT 26	INT 25	INT 24	INT 23	INT 22	INT 21	INT 20	INT 19	INT 18	INT 17	INT 16
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	19	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	INT 15	INT 14	INT 13	INT 12	INT 11	INT 10	INT 09	INT 08	INT 07	INT 06	INT 05	INT 04	INT 03	INT 02	INT 01	INT 00
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 24-14. eDMA Interrupt Request Register (EDMA_IRQRL)

Table 24-15. EDMA_IRQRL Field Descriptions

Field	Description
INT*n*	eDMA Interrupt Request *n.* 0 The interrupt request for channel *n* is cleared. 1 The interrupt request for channel *n* is active.

24.3.2.14 eDMA Error Register (EDMA_ERL)

The EDMA_ERL provides a bit map for the 32 channels signaling the presence of an error for each channel. EDMA_ERL maps to channels 31–0.

The DMA engine signals the occurrence of a error condition by setting the appropriate bit in this register. The outputs of this register are enabled by the contents of the EDMA_EEIR, then logically summed across 32 channels to form an error interrupt request, which is then routed to the interrupt controller. During the execution of the interrupt service routine associated with any eDMA errors, it is software's responsibility to clear the appropriate bit, negating the error interrupt request. Typically, a write to the EDMA_CER in the interrupt service routine is used for this purpose. The normal eDMA channel completion indicators, setting the transfer control descriptor DONE flag and the possible assertion of an interrupt request, are not affected when an error is detected.

The contents of this register can also be polled and a non-zero value indicates the presence of a channel error, regardless of the state of the EDMA_EEIR. The EDMA_ESR[VLD] bit is a logical OR of all bits in this register and it provides a single bit indication of any errors. The state of any given channel's error indicators is affected by writes to this register; it is also affected by writes to the EDMA_CER. On writes to EDMA_ERL, a 1 in any bit position clears the corresponding channel's error status. A 0 in any bit position has no effect on the corresponding channel's current error status. The EDMA_CER is provided so the error indicator for a single channel can be cleared.

Offset: EDMA_BASE + 0x002C Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	ERR 31	ERR 30	ERR 29	ERR 28	ERR 27	ERR 26	ERR 25	ERR 24	ERR 23	ERR 22	ERR 21	ERR 20	ERR 19	ERR 18	ERR 17	ERR 16
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	ERR 15	ERR 14	ERR 13	ERR 12	ERR 11	ERR 10	ERR 09	ERR 08	ERR 07	ERR 06	ERR 05	ERR 04	ERR 03	ERR 02	ERR 01	ERR 00
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 24-15. eDMA Error Register (EDMA_ERL)

Table 24-16. EDMA_ERL Field Descriptions

Field	Description
ERR*n*	eDMA Error *n.* 0 An error in channel *n* has not occurred. 1 An error in channel *n* has occurred.

24.3.2.15 DMA Hardware Request Status (EDMA_HRSL)

The EDMA_HRSL registers provide a bit map for the implemented channels (32) to show the current hardware request status for each channel. EDMA_HRSL covers channels 31–00.

See Table 24-17 for the EDMA_HRSL definition.

Address: EDMA_BASE + 0x0034 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	HRS 31	HRS 30	HRS 29	HRS 28	HRS 27	HRS 26	HRS 25	HRS 24	HRS 23	HRS 22	HRS 21	HRS 20	HRS 19	HRS 18	HRS 17	HRS 16
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	HRS 15	HRS 14	HRS 13	HRS 12	HRS 11	HRS 10	HRS 09	HRS 08	HRS 07	HRS 06	HRS 05	HRS 04	HRS 03	HRS 02	HRS 01	HRS 00
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 24-16. EDMA Hardware Request Status Register Low (EDMA_HRSL)

24.3.2.16 eDMA Channel *n* Priority Registers (EDMA_CPR*n*)

Table 24-17. EDMA_HRSL Field Descriptions

Field	Description
HRS*n*	DMA Hardware Request Status 0 A hardware service request for channel *n* is not present. 1 A hardware service request for channel *n* is present. **Note:** The hardware request status reflects the state of the request as seen by the arbitration logic. Therefore, this status is affected by the EDMA_ERQRL[ERQ*n*] bit.

When the fixed-priority channel arbitration mode is enabled (EDMA_CR[ERCA] = 0), the contents of these registers define the unique priorities associated with each channel. The channel priorities are evaluated by numeric value; that is, 0 is the lowest priority, 1 is the next higher priority, then 2, 3, etc. If software modifies channel priority values, then the software must ensure that the channel priorities contain unique values. Otherwise, a configuration error is reported. The range of the priority value is limited to the values of 0 through 31. When read, the GRPPRI bits of the EDMA_CPR*n* register reflect the current priority level of the group of channels in which the corresponding channel resides. GRPPRI bits are not affected by writes to the EDMA_CPR*n* registers. The group priority is assigned in the EDMA_CR. See Figure 24-2 and Table 24-3 for the EDMA_CR definition.

Channel preemption is enabled on a per-channel basis by setting the ECP bit in the EDMA_CPR*n* register. Channel preemption allows the executing channel's data transfers to be temporarily suspended in favor of starting a higher priority channel. After the preempting channel has completed all its minor loop data transfers, the preempted channel is restored and resumes execution. After the restored channel completes one read/write sequence, it is again eligible for preemption. If any higher priority channel requests service, the restored channel is suspended and the higher priority channel is serviced. Nested preemption (attempting to preempt a preempting channel) is not supported. After a preempting channel begins execution, it cannot be preempted. Preemption is available only when fixed arbitration is selected for both group and channel arbitration modes.

A channel's ability to preempt another channel can be disabled by setting EDMA_CPR[DPA]. When a channel's preempt ability is disabled, that channel cannot suspend a lower priority channel's data transfer; regardless of the lower priority channel's ECP setting. This allows for a pool of low priority, large data moving channels to be defined. These low priority channels can be configured to not preempt each other, thus preventing a low priority channel from consuming the preempt slot normally available a true, high priority channel.

Offset: EDMA_BASE + 0x0100 + *n* Access: User read/write

	0	1	2	3	4	5	6	7
R	ECP	DPA	GRPPRI		CHPRI			
W								
Reset	0	0	0	0	— [1]			

[1] The reset value for the channel priority field, CHPRI[0–3], is equal to the corresponding channel number for each priority register; that is, EDMA_CPRI0[CHPRI] = 0b0000 and EDMA_CPR15[CHPRI] = 0b1111.

Figure 24-17. eDMA Channel *n* Priority Register (EDMA_CPR*n*)

Table 24-18. EDMA_CPRn Field Descriptions

Field	Description
ECP	Enable Channel Preemption. 0 Channel *n* cannot be suspended by a higher priority channel's service request. 1 Channel *n* can be temporarily suspended by the service request of a higher priority channel.
DPA	Disable preempt ability. 0 Channel n can suspend a lower priority channel. 1 Channel n cannot suspend any channel, regardless of channel priority.
GRPPRI[0:1]	Channel *n* current group priority. Group priority assigned to this channel group when fixed-priority arbitration is enabled. These two bits are read-only; writes are ignored. The reset value for the group priority fields is equal to the corresponding channel number for each priority register; that is, EDMA_CPR31[GRPPRI] = 0b01.
CHPRI[0:3]	Channel *n* Arbitration Priority. Channel priority when fixed-priority arbitration is enabled. The reset value for the channel priority fields CHPRI[0–3], is equal to the corresponding channel number for each priority register; that is, EDMA_CPR31[CHPRI] = 0b1111.

24.3.2.17 Transfer Control Descriptor (TCD)

Each channel requires a 32-byte transfer control descriptor for defining the desired data movement operation. The channel descriptors are stored in the local memory in sequential order: channel 0, channel 1,... channel 31. The definitions of the TCD are presented as eight 32-bit values. Table 24-19 is a field list of the basic TCD structure.

Table 24-19. TCD*n* 32-bit Memory Structure

eDMA Offset	TCD*n* Field	
0x1000+(32 x *n*)+0x0000	Source address (saddr)	
0x1000+(32 x *n*)+0x0004	Transfer attributes	Signed source address offset (soff)
0x1000+(32 x *n*)+0x0008	Inner minor byte count (nbytes)	

Table 24-19. TCD*n* 32-bit Memory Structure (continued)

eDMA Offset	TCD*n* Field	
0x1000+(32 x *n*)+0x000C	Last source address adjustment (slast)	
0x1000+(32 x *n*)+0x0010	Destination address (daddr)	
0x1000+(32 x *n*)+0x0014	Current major iteration count (citer)	Signed destination address offset (doff)
0x1000 (32 x *n*) 0x0018	Last destination address adjustment / scatter-gather address (dlast_sga)	
0x1000+(32 x *n*)+0x001c	Beginning major iteration count (biter)	Channel control/status

Figure 24-18 and Table 24-20 define the fields of the TCD*n* structure.

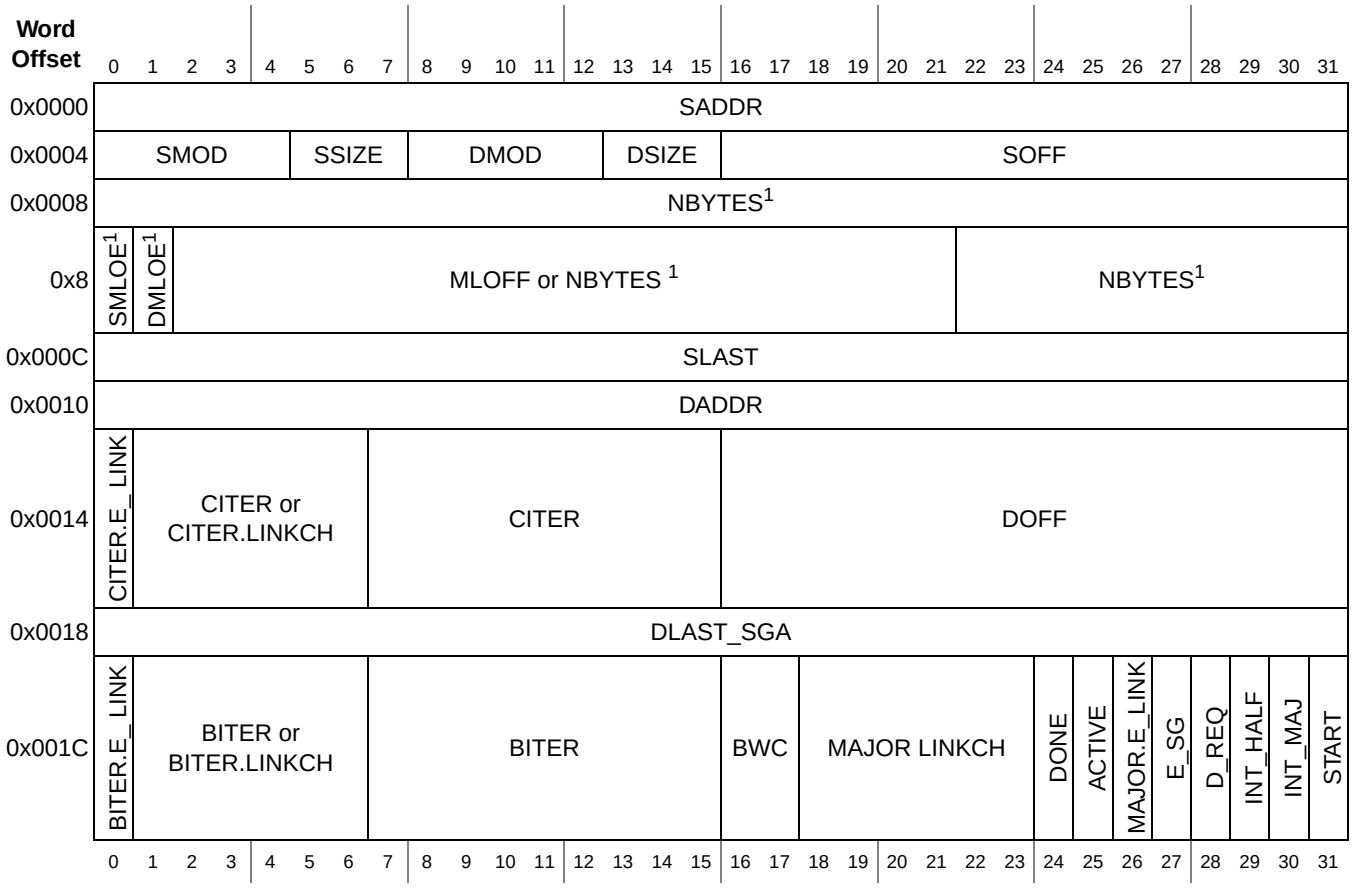


Figure 24-18. TCD Structure

1 The fields implemented in Word 2 depend on whether EDMA_CR(EMLM) is set to 0 or 1. Refer to Table 24-3.

NOTE

The TCD structures for the eDMA channels shown in Figure 24-18 are implemented in internal SRAM. These structures are not initialized at reset; therefore, all channel TCD parameters must be initialized by the application code before activating that channel.

Table 24-20. TCDn Field Descriptions

Bits / Word Offset [n:n]	Name	Description
0–31 / 0x0 [0:31]	SADDR [0:31]	Source address. Memory address pointing to the source data. Word 0x0, bits 0–31.
32–36 / 0x4 [0:4]	SMOD [0:4]	Source address modulo. 0 Source address modulo feature is disabled. non-0 This value defines a specific address range that is specified to be the value after SADDR + SOFF calculation is performed or the original register value. The setting of this field provides the ability to easily implement a circular data queue. For data queues requiring power-of-2 size bytes, the queue should start at a 0-modulo-size address and the SMOD field should be set to the appropriate value for the queue, freezing the desired number of upper address bits. The value programmed into this field specifies the number of lower address bits that are allowed to change. For this circular queue application, the SOFF is typically set to the transfer size to implement post-increment addressing with the SMOD function constraining the addresses to a 0-modulo-size range.
37–39 / 0x4 [5:7]	SSIZE [0:2]	Source data transfer size. 000 8-bit 001 16-bit 010 32-bit 011 64-bit 100 Reserved 101 32-byte (4-beat 64-bit burst) 110 Reserved 111 Reserved The attempted specification of a reserved encoding causes a configuration error.
40–44 / 0x4 [8:12]	DMOD [0:4]	Destination address modulo. See the SMOD[0:5] definition.
45–47 / 0x4 [13:15]	DSIZE [0:2]	Destination data transfer size. See the SSIZE[0:2] definition.
48–63 / 0x4 [16:31]	SOFF [0:15]	Source address signed offset. Sign-extended offset applied to the current source address to form the next-state value as each source read is completed.
64 0x8 [0]	SMLOE [1] 0	Source minor loop offset enable This flag selects whether the minor loop offset is applied to the source address upon minor loop completion. 0 The minor loop offset is not applied to the source address. 1 The minor loop offset is applied to the source address.
65 0x8 [1]	DMLOE [1] 1	Destination minor loop offset enable This flag selects whether the minor loop offset is applied to the destination address upon minor loop completion. 0 The minor loop offset is not applied to the destination address. 1 The minor loop offset is applied to the destination address.

Table 24-20. TCDn Field Descriptions (continued)

Bits / Word Offset [n:n]	Name	Description
66–85 0x8 [2-21]	MLOFF or NBYTES [1] [0:19]	Inner “minor” byte transfer count or Minor loop offset If both SMLOE and DMLOE are cleared, this field is part of the byte transfer count. If either SMLOE or DMLOE are set, this field represents a sign-extended offset applied to the source or destination address to form the next-state value after the minor loop is completed.
86–95 / 0x8 [22:31]	NBYTES [1]	Inner “minor” byte transfer count. Number of bytes to be transferred in each service request of the channel. As a channel is activated, the contents of the appropriate TCD is loaded into the eDMA engine, and the appropriate reads and writes performed until the complete byte transfer count has been transferred. This is an indivisible operation and cannot be stalled or halted. Once the minor count is exhausted, the current values of the SADDR and DADDR are written back into the local memory, the major iteration count is decremented and restored to the local memory. If the major iteration count is completed, additional processing is performed. **Note:** The NBYTES value of 0x0000_0000 is interpreted as 0x1_0000_0000, thus specifying a 4 GB transfer.
96–127 / 0xC [0:31]	SLAST [0:31]	Last source address adjustment. Adjustment value added to the source address at the completion of the outer major iteration count. This value can be applied to “restore” the source address to the initial value, or adjust the address to reference the next data structure.
128–159 / 0x10 [0:31]	DADDR [0:31]	Destination address. Memory address pointing to the destination data.
160 / 0x14 [0]	CITER.E_LINK	Enable channel-to-channel linking on minor loop completion. As the channel completes the inner minor loop, this flag enables the linking to another channel, defined by CITER.LINKCH[0:5]. The link target channel initiates a channel service request via an internal mechanism that sets the TCD.START bit of the specified channel. If channel linking is disabled, the CITER value is extended to 15 bits in place of a link channel number. If the major loop is exhausted, this link mechanism is suppressed in favor of the MAJOR.E_LINK channel linking. 0 The channel-to-channel linking is disabled. 1 The channel-to-channel linking is enabled. **Note:** This bit must be equal to the BITER.E_LINK bit. Otherwise, a configuration error is reported.
161–166 / 0x14 [1:6]	CITER [0:5] or CITER.LINKCH [0:5]	Current major iteration count or link channel number. If channel-to-channel linking is disabled (TCD.CITER.E_LINK = 0), then • No channel-to-channel linking (or chaining) is performed after the inner minor loop is exhausted. TCD bits [161:175] are used to form a 15-bit CITER field. Otherwise, • After the minor loop is exhausted, the DMA engine initiates a channel service request at the channel defined by CITER.LINKCH[0:5] by setting that channel’s TCD.START bit.

Table 24-20. TCDn Field Descriptions (continued)

Bits / Word Offset [n:n]	Name	Description
167–175 / 0x14 [7:15]	CITER [6:14]	Current major iteration count. This 9 or 15-bit count represents the current major loop count for the channel. It is decremented each time the minor loop is completed and updated in the transfer control descriptor memory. After the major iteration count is exhausted, the channel performs a number of operations (for example, final source and destination address calculations), optionally generating an interrupt to signal channel completion before reloading the CITER field from the beginning iteration count (BITER) field. **Note:** When the CITER field is initially loaded by software, it must be set to the same value as that contained in the BITER field. **Note:** If the channel is configured to execute a single service request, the initial values of BITER and CITER should be 0x0001.
176–191 / 0x14 [16:31]	DOFF [0:15]	Destination address signed Offset. Sign-extended offset applied to the current destination address to form the next-state value as each destination write is completed.
192–223 / 0x18 [0:31]	DLAST_SGA [0:31]	Last destination address adjustment or the memory address for the next transfer control descriptor to be loaded into this channel (scatter-gather). If scatter-gather processing for the channel is disabled (TCD.E_SG = 0) then • Adjustment value added to the destination address at the completion of the outer major iteration count. This value can be applied to restore the destination address to the initial value, or adjust the address to reference the next data structure. Otherwise, • This address points to the beginning of a 0-modulo-32 byte region containing the next transfer control descriptor to be loaded into this channel. This channel reload is performed as the major iteration count completes. The scatter-gather address must be 0-modulo-32 byte, otherwise a configuration error is reported.
224 / 0x1C [0]	BITER.E_LINK	Enables channel-to-channel linking on minor loop complete. As the channel completes the inner minor loop, this flag enables the linking to another channel, defined by BITER.LINKCH[0:5]. The link target channel initiates a channel service request via an internal mechanism that sets the TCD.START bit of the specified channel. If channel linking is disabled, the BITER value is extended to 15 bits in place of a link channel number. If the major loop is exhausted, this link mechanism is suppressed in favor of the MAJOR.E_LINK channel linking. 0 The channel-to-channel linking is disabled. 1 The channel-to-channel linking is enabled. **Note:** When the TCD is first loaded by software, this field must be set equal to the corresponding CITER field. Otherwise, a configuration error is reported. As the major iteration count is exhausted, the contents of this field is reloaded into the CITER field.

Table 24-20. TCDn Field Descriptions (continued)

Bits / Word Offset [n:n]	Name	Description
225–230 / 0x1C [1:6]	BITER [0:5] or BITER.LINKCH[0:5]	Starting major iteration count or link channel number. If channel-to-channel linking is disabled (TCD.BITER.E_LINK = 0), then • No channel-to-channel linking (or chaining) is performed after the inner minor loop is exhausted. TCD bits [225:239] are used to form a 15-bit BITER field. Otherwise, • After the minor loop is exhausted, the DMA engine initiates a channel service request at the channel, defined by BITER.LINKCH[0:5], by setting that channel's TCD.START bit. **Note:** When the TCD is first loaded by software, this field must be set equal to the corresponding CITER field. Otherwise, a configuration error is reported. As the major iteration count is exhausted, the contents of this field is reloaded into the CITER field.
231–239 / 0x1C [7:15]	BITER [6:14]	Starting major iteration count. As the transfer control descriptor is first loaded by software, this field must be equal to the value in the CITER field. As the major iteration count is exhausted, the contents of this field are reloaded into the CITER field. **Note:** If the channel is configured to execute a single service request, the initial values of BITER and CITER should be 0x0001.
240–241 / 0x1C [16:17]	BWC [0:1]	Bandwidth control. This two-bit field provides a mechanism to effectively throttle the amount of bus bandwidth consumed by the eDMA. In general, as the eDMA processes the inner minor loop, it continuously generates read/write sequences until the minor count is exhausted. This field forces the eDMA to stall after the completion of each read/write access to control the bus request bandwidth seen by the system bus crossbar switch (XBAR). 00 No DMA engine stalls 01 Reserved 10 DMA engine stalls for 4 cycles after each r/w 11 DMA engine stalls for 8 cycles after each r/w
242–247 / 0x1C [18:23]	MAJOR.LINKCH [0:5]	Link channel number. If channel-to-channel linking on major loop complete is disabled (TCD.MAJOR.E_LINK = 0) then, • No channel-to-channel linking (or chaining) is performed after the outer major loop counter is exhausted. Otherwise • After the major loop counter is exhausted, the DMA engine initiates a channel service request at the channel defined by MAJOR.LINKCH[0:5] by setting that channel's TCD.START bit.
248 / 0x1C [24]	DONE	Channel done. This flag indicates the eDMA has completed the outer major loop. It is set by the DMA engine as the CITER count reaches zero; it is cleared by software or hardware when the channel is activated (when the DMA engine has begun processing the channel, not when the first data transfer occurs). **Note:** This bit must be cleared to write the MAJOR.E_LINK or E_SG bits.
249 / 0x1C [25]	ACTIVE	Channel active. This flag signals the channel is currently in execution. It is set when channel service begins, and is cleared by the DMA engine as the inner minor loop completes or if any error condition is detected.

Table 24-20. TCDn Field Descriptions (continued)

Bits / Word Offset [n:n]	Name	Description
250 / 0x1C [26]	MAJOR.E_LINK	Enable channel-to-channel linking on major loop completion. As the channel completes the outer major loop, this flag enables the linking to another channel, defined by MAJOR.LINKCH[0:5]. The link target channel initiates a channel service request via an internal mechanism that sets the TCD.START bit of the specified channel. **Note:** To support the dynamic linking coherency model, this field is forced to zero when written to while the TCD.DONE bit is set. 0 The channel-to-channel linking is disabled. 1 The channel-to-channel linking is enabled.
251 / 0x1C [27]	E_SG	Enable scatter-gather processing. As the channel completes the outer major loop, this flag enables scatter-gather processing in the current channel. If enabled, the DMA engine uses DLAST_SGA as a memory pointer to a 0-modulo-32 address containing a 32-byte data structure which is loaded as the transfer control descriptor into the local memory. **Note:** To support the dynamic scatter-gather coherency model, this field is forced to zero when written to while the TCD.DONE bit is set. 0 The current channel's TCD is normal format. 1 The current channel's TCD specifies a scatter gather format. The DLAST_SGA field provides a memory pointer to the next TCD to be loaded into this channel after the outer major loop completes its execution.
252 / 0x1C [28]	D_REQ	Disable hardware request. If this flag is set, the eDMA hardware automatically clears the corresponding EDMA_ERQL bit when the current major iteration count reaches zero. 0 The channel's EDMA_ERQL bit is not affected. 1 The channel's EDMA_ERQL bit is cleared when the outer major loop is complete.
253 / 0x1C [29]	INT_HALF	Enable an interrupt when major counter is half complete. If this flag is set, the channel generates an interrupt request by setting the appropriate bit in the EDMA_IRQRL when the current major iteration count reaches the halfway point. Specifically, the comparison performed by the eDMA engine is (CITER == (BITER >> 1)). This halfway point interrupt request is provided to support double-buffered (aka ping-pong) schemes, or other types of data movement where the processor needs an early indication of the transfer's progress. CITER = BITER = 1 with INT_HALF enabled will generate an interrupt as it satisfies the equation (CITER == (BITER >> 1)) after a single activation. 0 The half-point interrupt is disabled. 1 The half-point interrupt is enabled.
254 / 0x1C [30]	INT_MAJ	Enable an interrupt when major iteration count completes. If this flag is set, the channel generates an interrupt request by setting the appropriate bit in the EDMA_ERQL when the current major iteration count reaches zero. 0 The end-of-major loop interrupt is disabled. 1 The end-of-major loop interrupt is enabled.
255 / 0x1C [31]	START	Channel start. If this flag is set the channel is requesting service. The eDMA hardware automatically clears this flag after the channel begins execution. 0 The channel is not explicitly started. 1 The channel is explicitly started via a software initiated service request.

24.4 Functional Description

This section provides an overview of the microarchitecture and functional operation of the eDMA block.

The eDMA module is partitioned into two major modules: the DMA engine and the transfer control descriptor local memory. The DMA engine is further partitioned into four submodules, which are detailed below.

- DMA engine
  - Address path: This module implements registered versions of two channel transfer control descriptors: channel x and channel y, and is responsible for all the master bus address calculations. All the implemented channels provide the same functionality. This hardware structure allows the data transfers associated with one channel to be preempted after the completion of a read/write sequence if a higher priority channel service request is asserted while the first channel is active. After a channel is activated, it runs until the minor loop is completed unless preempted by a higher priority channel. This capability provides a mechanism (optionally enabled by EDMA_CPR*n*[ECP]) where a large data move operation can be preempted to minimize the time another channel is blocked from execution.
  - When another channel is activated, the contents of its transfer control descriptor is read from the local memory and loaded into the registers of the other address path channel{x,y}. After the inner minor loop completes execution, the address path hardware writes the new values for the TCD*n*.{SADDR, DADDR, CITER} back into the local memory. If the major iteration count is exhausted, additional processing is performed, including the final address pointer updates, reloading the TCDn.CITER field, and a possible fetch of the next TCD*n* from memory as part of a scatter-gather operation.
  - Data path: This module implements the actual bus master read/write datapath. It includes 32 bytes of register storage (matching the maximum transfer size) and the necessary mux logic to support any required data alignment. The system read data bus is the primary input, and the system write data bus is the primary output.
  - The address and data path modules directly support the two-stage pipelined system bus. The address path module represents the 1st stage of the bus pipeline (the address phase), while the data path module implements the second stage of the pipeline (the data phase).
  - Program model/channel arbitration: This module implements the first section of eDMA's programming model and also the channel arbitration logic. The programming model registers are connected to the slave bus (not shown). The eDMA peripheral request inputs and eDMA interrupt request outputs are also connected to this module (via the control logic).
  - Control: This module provides all the control functions for the DMA engine. For data transfers where the source and destination sizes are equal, the DMA engine performs a series of source read, destination write operations until the number of bytes specified in the inner minor loop byte count has been moved.

    A minor loop interaction is defined as the number of bytes to transfer (*n*bytes) divided by the transfer size. Transfer size is defined as:

    if (SSIZE < DSIZE)

    transfer size = destination transfer size (# of bytes)

    else

    transfer size = source transfer size (# of bytes)

Minor loop TCD variables are SOFF, SMOD, DOFF, DMOD, NBYTES, SADDR, DADDR, BWC, ACTIVE, AND START. Major loop TCD variables are DLAST, SLAST, CITER, BITER, DONE, D_REQ, INT_MAJ, MAJOR_LNKCH, and INT_HALF.

For descriptors where the sizes are not equal, multiple access of the smaller size data are required for each reference of the larger size. For example, if the source size references 16-bit data and the destination is 32-bit data, two reads are performed, then one 32-bit write.

- TCD local memory
  - Memory controller: This logic implements the required dual-ported controller, handling accesses from both the DMA engine as well as references from the slave bus. As noted earlier, in the event of simultaneous accesses, the DMA engine is given priority and the slave transaction is stalled.
  - Memory array: The TCD is implemented using a single-ported, synchronous compiled RAM memory array.

24.4.1 eDMA Basic Data Flow

The eDMA transfers data based on a two-deep, nested flow. The basic flow of a data transfer can be partitioned into three segments. As shown in Figure 24-19, the first segment involves the channel service request. In the diagram, this example uses the assertion of the eDMA peripheral request signal to request service for channel *n*. Channel service request via software and the TCDn.START bit follows the same basic flow as an eDMA peripheral request. The eDMA peripheral request input signal is registered internally and then routed to through the DMA engine, first through the control module, then into the program model/channel arbitration module. In the next cycle, the channel arbitration is performed using the fixed-priority or round-robin algorithm. After the arbitration is complete, the activated channel number is sent through the address path and converted into the required address to access the TCD local memory. Next, the TCD memory is accessed and the required descriptor read from the local memory and loaded into the DMA engine address path channel{x,y} registers. The TCD memory is organized 64-bits in width to minimize the time needed to fetch the activated channel's descriptor and load it into the eDMA engine address path channel{x,y} registers.

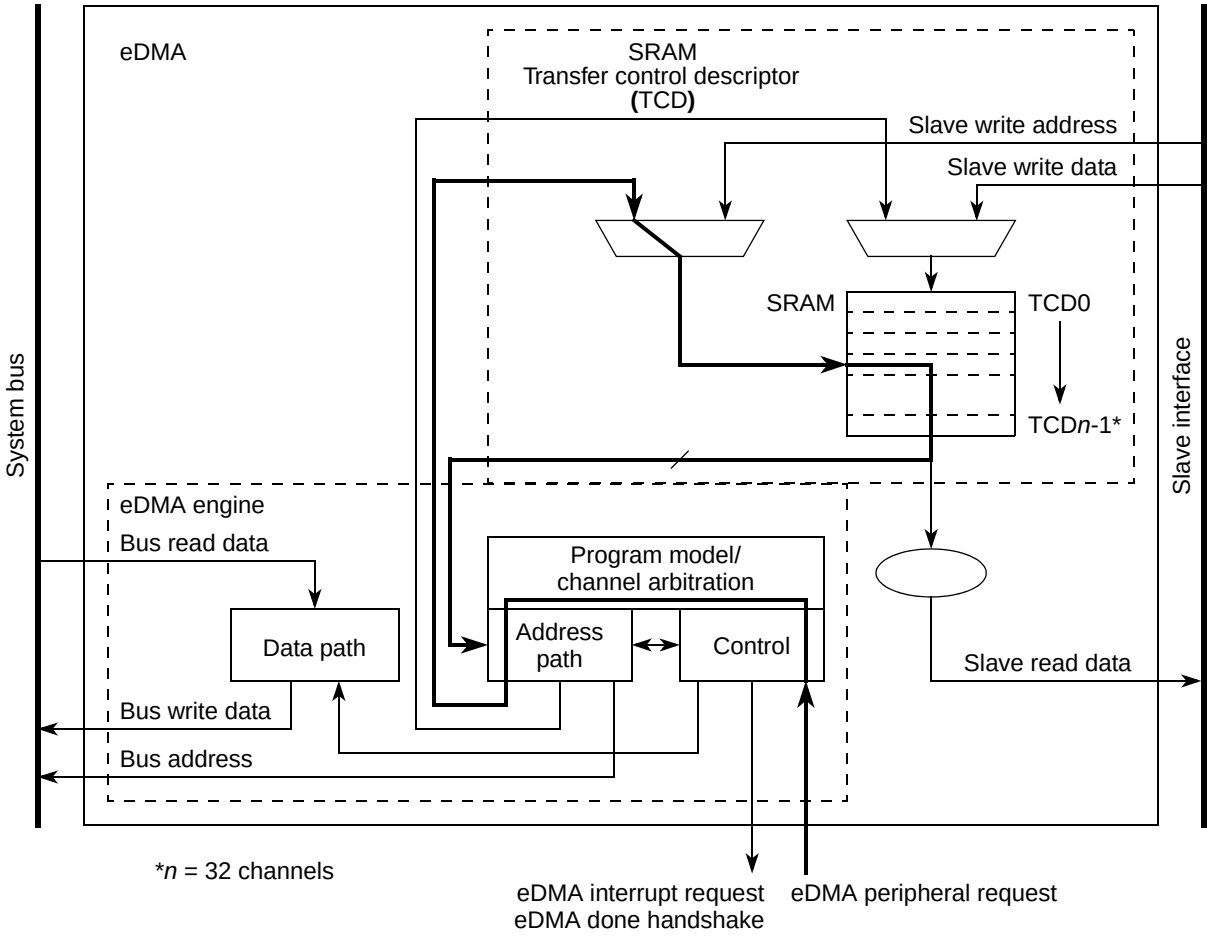


Figure 24-19. eDMA Operation, Part 1

In the second part of the basic data flow as shown in Figure 24-20, the modules associated with the data transfer (address path, data path, and control) sequence through the required source reads and destination writes to perform the actual data movement. The source reads are initiated and the fetched data is temporarily stored in the data path module until it is gated onto the system bus during the destination write. This source read/destination write processing continues until the inner minor byte count has been transferred. The eDMA done handshake signal is asserted at the end of the minor byte count transfer.

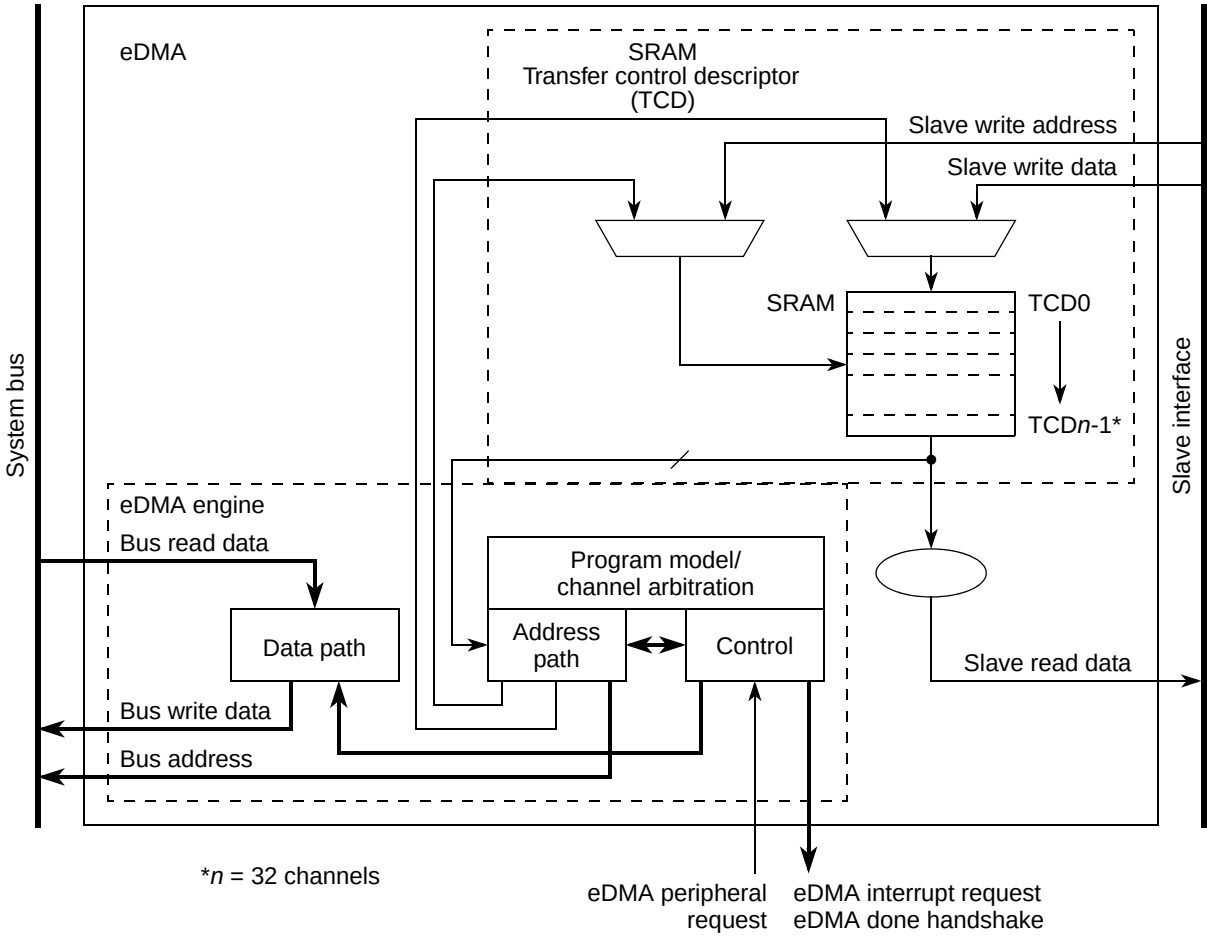


Figure 24-20. eDMA Operation, Part 2

After the inner minor byte count has been moved, the final phase of the basic data flow is performed. In this segment, the address path logic performs the required updates to certain fields in the channel's TCD; for example, SADDR, DADDR, CITER. If the outer major iteration count is exhausted, then there are additional operations performed. These include the final address adjustments and reloading of the BITER field into the CITER. Additionally, assertion of an optional interrupt request occurs at this time, as does a possible fetch of a new TCD from memory using the scatter-gather address pointer included in the descriptor. The updates to the TCD memory and the assertion of an interrupt request are shown in Figure 24-21.

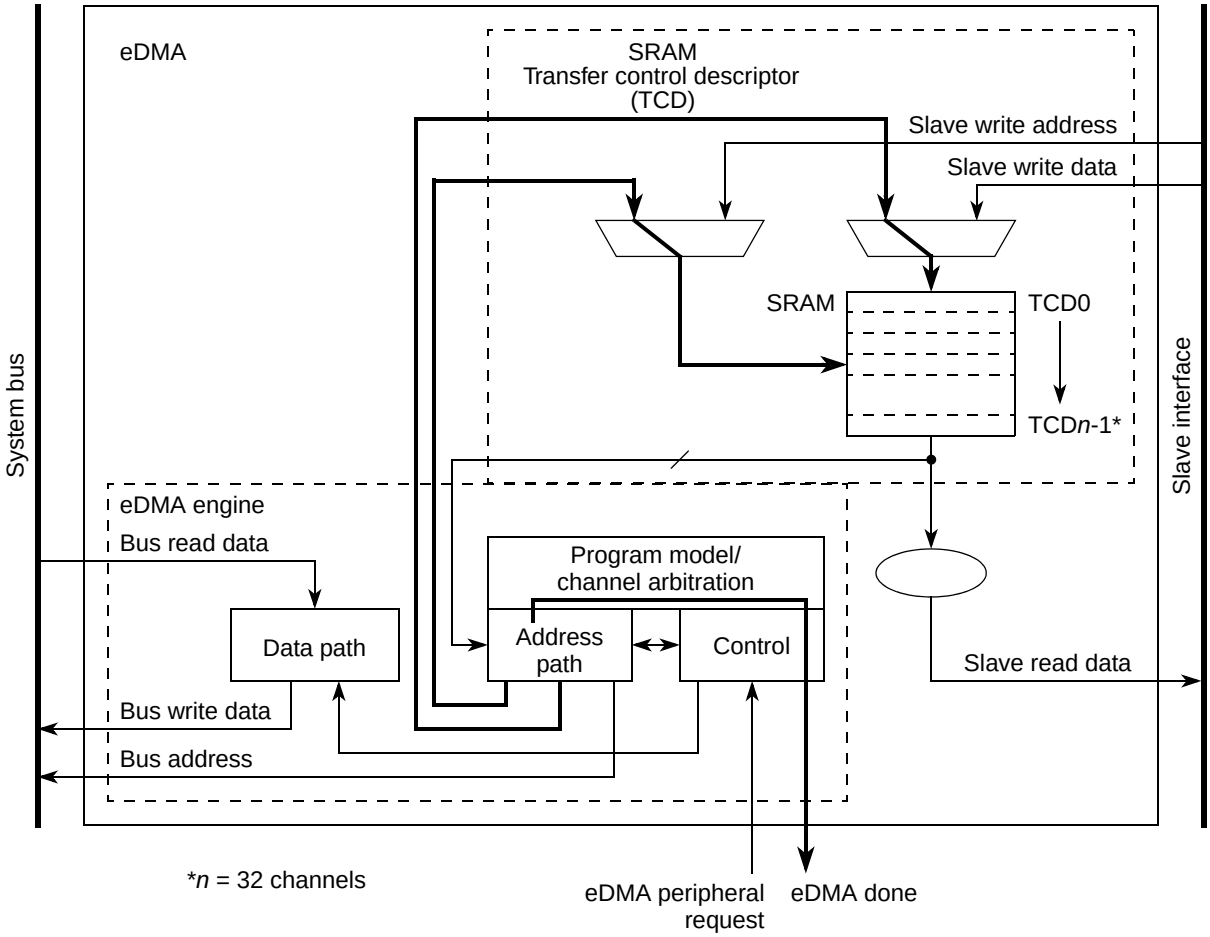


Figure 24-21. eDMA Operation, Part 3

24.5 Initialization / Application Information

24.5.1 eDMA Initialization

A typical initialization of the eDMA has the following sequence:

1. Write the EDMA_CR if a configuration other than the default is desired.
2. Write the channel priority levels into the EDMA_CPR*n* registers if a configuration other than the default is desired.
3. Enable error interrupts in the EDMA_EEIRL and/or EDMA_EEIRH registers if desired.
4. Write the 32-byte TCD for each channel that may request service.
5. Enable any hardware service requests via the EDMA_ERQRH and/or EDMA_ERQRL registers.
6. Request channel service by software (setting the TCD.START bit) or by hardware (slave device asserting its DMA peripheral request signal).

After any channel requests service, a channel is selected for execution based on the arbitration and priority levels written into the programmer's model. The DMA engine reads the entire TCD, including the primary

transfer control parameter shown in Table 24-21, for the selected channel into its internal address path module. As the TCD is being read, the first transfer is initiated on the system bus unless a configuration error is detected. Transfers from the source (as defined by the source address, TCD.SADDR) to the destination (as defined by the destination address, TCD.DADDR) continue until the specified number of bytes (TCD.NBYTES) have been transferred. When the transfer is complete, the DMA engine's local TCD.SADDR, TCD.DADDR, and TCD.CITER are written back to the main TCD memory and any minor loop channel linking is performed, if enabled. If the major loop is exhausted, further post processing is executed; for example, interrupts, major loop channel linking, and scatter-gather operations, if enabled.

Table 24-21. TCD Primary Control and Status Fields

TCD Field Name	Description
START	Control bit to start channel when using a software initiated DMA service (Automatically cleared by hardware)
ACTIVE	Status bit indicating the channel is currently in execution
DONE	Status bit indicating major loop completion (cleared by software when using a software initiated DMA service)
D_REQ	Control bit to disable DMA request at end of major loop completion when using a hardware-initiated DMA service
BWC	Control bits for throttling bandwidth control of a channel
E_SG	Control bit to enable scatter-gather feature
INT_HALF	Control bit to enable interrupt when major loop is half complete
INT_MAJ	Control bit to enable interrupt when major loop completes

Figure 24-22 shows how each DMA request initiates one minor loop transfer (iteration) without CPU intervention. DMA arbitration can occur after each minor loop, and one level of minor loop DMA preemption is allowed. The number of minor loops in a major loop is specified by the beginning iteration count (biter).

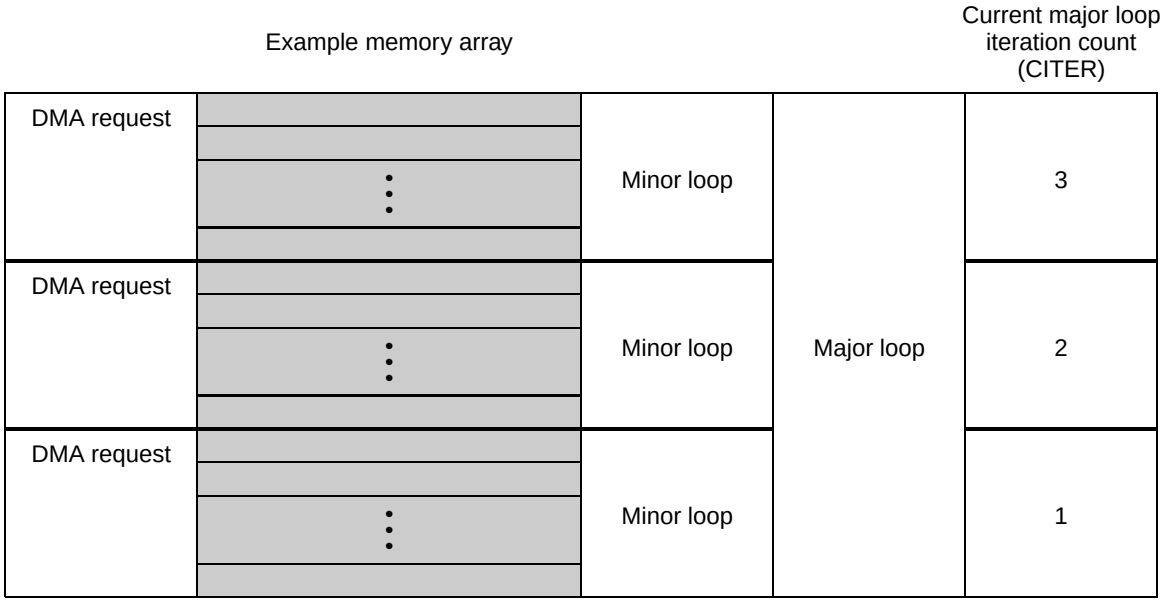


Figure 24-22. Example of Multiple Loop Iterations

Figure 24-23 lists the memory array terms and how the TCD settings interrelate.

xADDR: (Starting address)	xSIZE: (Size of one data transfer)	Minor loop (NBYTES in minor loop, often the same value as xSIZE)	Offset (xOFF): Number of bytes added to current address after each transfer (Often the same value as xSIZE) Each DMA source (S) and destination (D) has its own: • Address (xADDR) • Size (xSIZE) • Offset (xOFF) • Modulo (xMOD) • Last address adjustment (xLAST) where x = S or D Peripheral queues typically have size and offset equal to NBYTES
• • •	• • •	Minor loop	
xLAST: Number of bytes added to current address after major loop (typically used to loop back)	• • •	Last minor loop	

Figure 24-23. Memory Array Terms

24.5.2 DMA Programming Errors

The DMA performs various tests on the transfer control descriptor to verify consistency in the descriptor data. Most programming errors are reported on a per-channel basis with the exception of two errors: group-priority error and channel-priority error, or EDMA_ESR[GPE] and EDMA_ESR[CPE], respectively.

For all error types other than group- or channel-priority errors, the channel number causing the error is recorded in the EDMA_ESR. If the error source is not removed before the next activation of the problem channel, the error is detected and recorded again.

Channel-priority errors are identified within a group after that group has been selected as the active group. For the example, all of the channel priorities in group 1 are unique, but some of the channel priorities in group 0 are the same:

1. The DMA is configured for fixed-group and fixed-channel arbitration modes.
2. Group 1 is the highest priority and all channels are unique in that group.
3. Group 0 is the next highest priority and has two channels with the same priority level.
4. If group 1 has any service requests, those requests are executed.
5. After all of group 1 requests have completed, group 0 becomes the next active group.
6. If group 0 has a service request, then an undefined channel in group 0 is selected and a channel-priority error will occur.
7. This repeats until the all of group 0 requests have been removed or a higher priority group 1 request comes in.

In this sequence, for item 2, the DMA acknowledge lines assert only if the selected channel is requesting service via the DMA peripheral request signal. If interrupts are enabled for all channels, the user receives an error interrupt, but the channel number for the EDMA_ER and the error interrupt request line are undetermined because they reflect the undefined channel. A group-priority error is global and any request in any group causes a group-priority error.

If priority levels are not unique, the highest (channel/group) priority that has an active request is selected, but the lowest numbered (channel/group) with that priority is selected by arbitration and executed by the DMA engine. The hardware service request handshake signals, error interrupts, and error reporting are associated with the selected channel.

24.5.3 DMA Request Assignments

The assignments between the DMA requests from the modules to the channels of the eDMA are shown in Table 24-22. The source column is written in C language syntax. The syntax is module_instance.register[bit].

Table 24-22. DMA Request Summary for eDMA

DMA Request	Channel	Source	Description
DMA_MUX_CHCONFIG0_SOURCE	0	DMA_MUX.CHCONFIG0[SOURCE]	DMA MUX channel 0 source
DMA_MUX_CHCONFIG1_SOURCE	1	DMA_MUX.CHCONFIG1[SOURCE]	DMA MUX channel 1 source
DMA_MUX_CHCONFIG2_SOURCE	2	DMA_MUX.CHCONFIG2[SOURCE]	DMA MUX channel 2 source
DMA_MUX_CHCONFIG3_SOURCE	3	DMA_MUX.CHCONFIG3[SOURCE]	DMA MUX channel 3 source
DMA_MUX_CHCONFIG4_SOURCE	4	DMA_MUX.CHCONFIG4[SOURCE]	DMA MUX channel 4 source
DMA_MUX_CHCONFIG5_SOURCE	5	DMA_MUX.CHCONFIG5[SOURCE]	DMA MUX channel 5 source
DMA_MUX_CHCONFIG6_SOURCE	6	DMA_MUX.CHCONFIG6[SOURCE]	DMA MUX channel 6 source
DMA_MUX_CHCONFIG7_SOURCE	7	DMA_MUX.CHCONFIG7[SOURCE]	DMA MUX channel 7 source
DMA_MUX_CHCONFIG8_SOURCE	8	DMA_MUX.CHCONFIG8[SOURCE]	DMA MUX channel 8 source

Table 24-22. DMA Request Summary for eDMA (continued)

DMA Request	Channel	Source	Description
DMA_MUX_CHCONFIG9_SOURCE	9	DMA_MUX.CHCONFIG9[SOURCE]	DMA MUX channel 9 source
DMA_MUX_CHCONFIG10_SOURCE	10	DMA_MUX.CHCONFIG10[SOURCE]	DMA MUX channel 10 source
DMA_MUX_CHCONFIG11_SOURCE	11	DMA_MUX.CHCONFIG11[SOURCE]	DMA MUX channel 11 source
DMA_MUX_CHCONFIG12_SOURCE	12	DMA_MUX.CHCONFIG12[SOURCE]	DMA MUX channel 12 source
DMA_MUX_CHCONFIG13_SOURCE	13	DMA_MUX.CHCONFIG13[SOURCE]	DMA MUX channel 13 source
DMA_MUX_CHCONFIG14_SOURCE	14	DMA_MUX.CHCONFIG14[SOURCE]	DMA MUX channel 14 source
DMA_MUX_CHCONFIG15_SOURCE	15	DMA_MUX.CHCONFIG15[SOURCE]	DMA MUX channel 15 source
DMA_MUX_CHCONFIG16_SOURCE	16	DMA_MUX.CHCONFIG16[SOURCE]	DMA MUX channel 16 source
DMA_MUX_CHCONFIG17_SOURCE	17	DMA_MUX.CHCONFIG17[SOURCE]	DMA MUX channel 17 source
DMA_MUX_CHCONFIG18_SOURCE	18	DMA_MUX.CHCONFIG18[SOURCE]	DMA MUX channel 18 source
DMA_MUX_CHCONFIG19_SOURCE	19	DMA_MUX.CHCONFIG19[SOURCE]	DMA MUX channel 19 source
DMA_MUX_CHCONFIG20_SOURCE	20	DMA_MUX.CHCONFIG20[SOURCE]	DMA MUX channel 20 source
DMA_MUX_CHCONFIG21_SOURCE	21	DMA_MUX.CHCONFIG21[SOURCE]	DMA MUX channel 21 source
DMA_MUX_CHCONFIG22_SOURCE	22	DMA_MUX.CHCONFIG22[SOURCE]	DMA MUX channel 22 source
DMA_MUX_CHCONFIG23_SOURCE	23	DMA_MUX.CHCONFIG23[SOURCE]	DMA MUX channel 23 source
DMA_MUX_CHCONFIG24_SOURCE	24	DMA_MUX.CHCONFIG24[SOURCE]	DMA MUX channel 24 source
DMA_MUX_CHCONFIG25_SOURCE	25	DMA_MUX.CHCONFIG25[SOURCE]	DMA MUX channel 25 source
DMA_MUX_CHCONFIG26_SOURCE	26	DMA_MUX.CHCONFIG26[SOURCE]	DMA MUX channel 26 source
DMA_MUX_CHCONFIG27_SOURCE	27	DMA_MUX.CHCONFIG27[SOURCE]	DMA MUX channel 27 source
DMA_MUX_CHCONFIG28_SOURCE	28	DMA_MUX.CHCONFIG28[SOURCE]	DMA MUX channel 28 source
DMA_MUX_CHCONFIG29_SOURCE	29	DMA_MUX.CHCONFIG29[SOURCE]	DMA MUX channel 29 source
DMA_MUX_CHCONFIG30_SOURCE	30	DMA_MUX.CHCONFIG30[SOURCE]	DMA MUX channel 30 source
DMA_MUX_CHCONFIG31_SOURCE	31	DMA_MUX.CHCONFIG31[SOURCE]	DMA MUX channel 31 source

24.5.4 DMA Arbitration Mode Considerations

24.5.4.1 Fixed-Group Arbitration, Fixed-Channel Arbitration

In this mode, the channel service request from the highest priority channel in the highest priority group is selected to execute. If the eDMA is programmed so the channels within one group use fixed priorities, and that group is assigned the highest fixed priority of all groups, it is possible for that group to take all the bandwidth of the eDMA controller. That is, no other groups can be serviced if there is always at least one DMA request pending on a channel in the highest priority group when the controller arbitrates the next DMA request. The advantage of this scenario is that latency can be small for channels that need to be serviced quickly. Preemption is available in this scenario only.

24.5.4.2 Round-Robin Group Arbitration, Fixed-Channel Arbitration

When one or more DMA requests arrive from one or more groups, the channel with the highest priority from a specific group is serviced first. Groups are serviced starting with the highest group number with a service request and rotating through to the lowest group number containing a service request.

After the channel request is serviced, the group round robin algorithm selects the highest pending request from the next group in the round-robin sequence. Servicing continues round robin, always servicing the highest priority channel in the next group in the sequence, or skipping a group if it has no pending requests.

If a channel requests service at a rate that equals or exceeds the round robin service rate, then that channel is always serviced before lower priority channels in the same group, and the lower priority channels are never serviced. The advantage of this scenario is that no one group can consume all the eDMA bandwidth. The highest priority channel selection latency is potentially greater than fixed/fixed arbitration. Excessive request rates on high-priority channels can prevent the servicing of lower priority channels in the same group.

24.5.4.3 Round-Robin Group Arbitration, Round-Robin Channel Arbitration

Groups are serviced as described in Section 24.5.4.2, Round-Robin Group Arbitration, Fixed-Channel Arbitration, but this time channels are serviced in channel number order. One channel only is serviced from each requesting group for each round robin pass through the groups.

Within each group, channels are serviced starting with the highest channel number and rotating through to the lowest channel number without regard to channel priority levels.

Because channels are serviced in round-robin manner, any channel that generates DMA requests faster than a combination of the group round-robin service rate and the channel service rate for its group does not prevent the servicing of other channels in its group. Any DMA requests that are not serviced are simply lost, but at least one channel gets serviced.

This scenario ensures that all channels are guaranteed service at some point, regardless of the request rates. However, the potential latency could be high. All channels are treated equally. Priority levels are not used in round-robin/round-robin mode.

24.5.4.4 Fixed-Group Arbitration, Round-Robin Channel Arbitration

The highest priority group with a request is serviced. Lower priority groups are serviced if no pending requests exist in the higher priority groups.

Within each group, channels are serviced starting with the highest channel number and rotating through to the lowest channel number without regard to the channel priority levels assigned within the group.

This scenario could cause the same bandwidth consumption problem as indicated in Section 24.5.4.1, Fixed-Group Arbitration, Fixed-Channel Arbitration, but all the channels in the highest priority group get serviced. Service latency is short on the highest priority group, but could potentially get longer and longer as the group priority decreases.

24.5.5 DMA Transfer

24.5.5.1 Single Request

To perform a simple transfer of *n* bytes of data with one activation, set the major loop to 1 (TCD.CITER = TCD.BITER = 1). The data transfer begins after the channel service request is acknowledged and the channel is selected to execute. After the transfer is complete, the TCD.DONE bit is set and an interrupt is generated if properly enabled.

For example, the following TCD entry is configured to transfer 16 bytes of data. The eDMA is programmed for one iteration of the major loop transferring 16 bytes per iteration. The source memory has a byte wide memory port located at 0x1000. The destination memory has a word wide port located at 0x2000. The address offsets are programmed in increments to match the size of the transfer; one byte for the source and four bytes for the destination. The final source and destination addresses are adjusted to return to their beginning values.

TCD.CITER = TCD.BITER = 1
TCD.NBYTES = 16
TCD.SADDR = 0x1000
TCD.SOFF = 1
TCD.SSIZE = 0
TCD.SLAST = –16
TCD.DADDR = 0x2000
TCD.DOFF = 4
TCD.DSIZE = 2
TCD.DLAST_SGA= -16
TCD.INT_MAJ = 1
TCD.START = 1 (Must be written last after all other fields have been initialized)
All other TCD fields = 0

This would generate the following sequence of events:

1. Slave write to the TCD.START bit requests channel service.
2. The channel is selected by arbitration for servicing.
3. eDMA engine writes: TCD.DONE = 0, TCD.START = 0, TCD.ACTIVE = 1.
4. eDMA engine reads: channel TCD data from local memory to internal register file.
5. The source to destination transfers are executed as follows:
   a) read_byte(0x1000), read_byte(0x1001), read_byte(0x1002), read_byte(0x1003)
   b) write_word(0x2000) → first iteration of the minor loop
   c) read_byte(0x1004), read_byte(0x1005), read_byte(0x1006), read_byte(0x1007)
   d) write_word(0x2004) → second iteration of the minor loop
   e) read_byte(0x1008), read_byte(0x1009), read_byte(0x100a), read_byte(0x100b)
   f) write_word(0x2008) → third iteration of the minor loop

g) read_byte(0x100c), read_byte(0x100d), read_byte(0x100e), read_byte(0x100f)
h) write_word(0x200c) → last iteration of the minor loop → major loop complete

6. eDMA engine writes: TCD.SADDR = 0x1000, TCD.DADDR = 0x2000, TCD.CITER = 1 (TCD.BITER).
7. eDMA engine writes: TCD.ACTIVE = 0, TCD.DONE = 1, EDMA_IRQR*n* = 1.
8. The channel retires.

The eDMA goes idle or services the next channel.

24.5.5.2 Multiple Requests

The next example is the same as previous, excepting transferring 32 bytes via two hardware requests. The only fields that change are the major loop iteration count and the final address offsets. The eDMA is programmed for two iterations of the major loop transferring 16 bytes per iteration. After the channel's hardware requests are enabled in the EDMA_ERQR, channel service requests are initiated by the slave device (ERQR should be set after TCD). Note that TCD.START = 0.

TCD.CITER = TCD.BITER = 2
TCD.NBYTES = 16
TCD.SADDR = 0x1000
TCD.SOFF = 1
TCD.SSIZE = 0
TCD.SLAST = –32
TCD.DADDR = 0x2000
TCD.DOFF = 4
TCD.DSIZE = 2
TCD.DLAST_SGA= –32
TCD.INT_MAJ = 1
TCD.START = 0 (Must be written last after all other fields have been initialized)
All other TCD fields = 0

This generates the following sequence of events:

1. First hardware (eDMA peripheral request) request for channel service.
2. The channel is selected by arbitration for servicing.
3. eDMA engine writes: TCD.DONE = 0, TCD.START = 0, TCD.ACTIVE = 1.
4. eDMA engine reads: channel TCD data from local memory to internal register file.
5. The source to destination transfers are executed as follows:
   a) read_byte(0x1000), read_byte(0x1001), read_byte(0x1002), read_byte(0x1003)
   b) write_word(0x2000) → first iteration of the minor loop
   c) read_byte(0x1004), read_byte(0x1005), read_byte(0x1006), read_byte(0x1007)
   d) write_word(0x2004) → second iteration of the minor loop

e) read_byte(0x1008), read_byte(0x1009), read_byte(0x100a), read_byte(0x100b)
f) write_word(0x2008) → third iteration of the minor loop
g) read_byte(0x100c), read_byte(0x100d), read_byte(0x100e), read_byte(0x100f)
h) write_word(0x200c) → last iteration of the minor loop

6. eDMA engine writes: TCD.SADDR = 0x1010, TCD.DADDR = 0x2010, TCD.CITER = 1.
7. eDMA engine writes: TCD.ACTIVE = 0.
8. The channel retires → one iteration of the major loop.

The eDMA goes idle or services the next channel.

9. Second hardware (eDMA peripheral request) requests channel service.
10. The channel is selected by arbitration for servicing.
11. eDMA engine writes: TCD.DONE = 0, TCD.START = 0, TCD.ACTIVE = 1.
12. eDMA engine reads: channel TCD data from local memory to internal register file.
13. The source to destination transfers are executed as follows:
    a) read_byte(0x1010), read_byte(0x1011), read_byte(0x1012), read_byte(0x1013)
    b) write_word(0x2010) → first iteration of the minor loop
    c) read_byte(0x1014), read_byte(0x1015), read_byte(0x1016), read_byte(0x1017)
    d) write_word(0x2014) → second iteration of the minor loop
    e) read_byte(0x1018), read_byte(0x1019), read_byte(0x101a), read_byte(0x101b)
    f) write_word(0x2018) → third iteration of the minor loop
    g) read_byte(0x101c), read_byte(0x101d), read_byte(0x101e), read_byte(0x101f)
    h) write_word(0x201c) → last iteration of the minor loop → major loop complete
14. eDMA engine writes: TCD.SADDR = 0x1000, TCD.DADDR = 0x2000, TCD.CITER = 2 (TCD.BITER).
15. eDMA engine writes: TCD.ACTIVE = 0, TCD.DONE = 1, EDMA_IRQR*n* = 1.
16. The channel retires → major loop complete.

The eDMA goes idle or services the next channel.

24.5.5.3 Modulo Feature

The modulo feature of the eDMA provides the ability to implement a circular data queue in which the size of the queue is a power of two. MOD is a 5-bit bitfield for both the source and destination in the TCD and specifies which lower address bits are allowed to increment from their original value after the address + offset calculation. All upper address bits remain the same as in the original value. A setting of 0 for this field disables the modulo feature.

Table 24-23 shows how the transfer addresses are specified based on the setting of the MOD field. Here a circular buffer is created where the address wraps to the original value while the 28 upper address bits (0x1234567x) retain their original value. In this example the source address is set to 0x12345670, the offset is set to 4 bytes and the mod field is set to 4, allowing for a 2^4 byte (16-byte) size queue.

Table 24-23. Modulo Feature Example

Transfer Number	Address
1	0x12345670
2	0x12345674
3	0x12345678
4	0x1234567C
5	0x12345670
6	0x12345674

24.5.6 TCD Status

24.5.6.1 Minor Loop Complete

There are two methods to test for minor loop completion when using software initiated service requests. The first method is to read the TCD.CITER field and test for a change. Another method may be extracted from the sequence below. The second method is to test the TCD.START bit AND the TCD.ACTIVE bit. The minor loop complete condition is indicated by both bits reading zero after the TCD.START was written to a 1. Polling the TCD.ACTIVE bit may be inconclusive because the active status may be missed if the channel execution is short in duration.

The TCD status bits execute the following sequence for a software activated channel:

1. TCD.START = 1, TCD.ACTIVE = 0, TCD.DONE = 0 (channel service request via software).
2. TCD.START = 0, TCD.ACTIVE = 1, TCD.DONE = 0 (channel is executing).
3. TCD.START = 0, TCD.ACTIVE = 0, TCD.DONE = 0 (channel has completed the minor loop and is idle), or
4. TCD.START = 0, TCD.ACTIVE = 0, TCD.DONE = 1 (channel has completed the major loop and is idle).

The best method to test for minor loop completion when using hardware initiated service requests is to read the TCD.CITER field and test for a change. The hardware request and acknowledge handshakes signals are not visible in the programmer’s model.

The TCD status bits execute the following sequence for a hardware activated channel:

1. eDMA peripheral request asserts (channel service request via hardware).
2. TCD.START = 0, TCD.ACTIVE = 1, TCD.DONE = 0 (channel is executing).
3. TCD.START = 0, TCD.ACTIVE = 0, TCD.DONE = 0 (channel has completed the minor loop and is idle), or
4. TCD.START = 0, TCD.ACTIVE = 0, TCD.DONE = 1 (channel has completed the major loop and is idle).

For both activation types, the major loop complete status is explicitly indicated via the TCD.DONE bit.

The TCD.START bit is cleared automatically when the channel begins execution, regardless of how the channel was activated.

24.5.6.2 Active Channel TCD Reads

The eDMA will read back the true TCD.SADDR, TCD.DADDR, and TCD.NBYTES values if read while a channel is executing. The true values of the SADDR, DADDR, and NBYTES are the values the eDMA engine is currently using in its internal register file and not the values in the TCD local memory for that channel. The addresses (SADDR and DADDR) and NBYTES (decrements to zero as the transfer progresses) can give an indication of the progress of the transfer. All other values are read back from the TCD local memory.

24.5.6.3 Preemption Status

Preemption is available only when fixed arbitration is selected for both group- and channel-arbitration modes. A preempt-able situation is one in which a preempt-enabled channel is running and a higher priority request becomes active. When the eDMA engine is not operating in fixed group, fixed-channel arbitration mode, the determination of the relative priority of the actively running and the outstanding requests become undefined. Channel and group priorities are treated as equal (or more exactly, constantly rotating) when round-robin arbitration mode is selected.

The TCD.ACTIVE bit for the preempted channel remains asserted throughout the preemption. The preempted channel is temporarily suspended while the preempting channel executes one iteration of the major loop. Two TCD.ACTIVE bits set at the same time in the overall TCD map indicates a higher priority channel is actively preempting a lower priority channel.

24.5.7 Channel Linking

Channel linking (or chaining) is a mechanism in which one channel sets the TCD.START bit of another channel (or itself), thus initiating a service request for that channel. This operation is automatically performed by the eDMA engine at the conclusion of the major or minor loop when properly enabled.

The minor loop channel linking occurs at the completion of the minor loop (or one iteration of the major loop). The TCD.CITER.E_LINK field are used to determine whether a minor loop link is requested. When enabled, the channel link is made after each iteration of the minor loop except for the last. When the major loop is exhausted, only the major loop channel link fields are used to determine if a channel link should be made. For example, with the initial fields of:

TCD.CITER.E_LINK = 1
TCD.CITER.LINKCH = 0xC
TCD.CITER value = 0x4
TCD.MAJOR.E_LINK = 1
TCD.MAJOR.LINKCH = 0x7

will execute as:

1. Minor loop done → set channel 12 TCD.START bit
2. Minor loop done → set channel 12 TCD.START bit

3. Minor loop done → set channel 12 TCD.START bit
4. Minor loop done, major loop done → set channel 7 TCD.START bit

When minor loop linking is enabled (TCD.CITER.E_LINK = 1), the TCD.CITER field uses a nine bit vector to form the current iteration count.

When minor loop linking is disabled (TCD.CITER.E_LINK = 0), the TCD.CITER field uses a 15-bit vector to form the current iteration count. The bits associated with the TCD.CITER.LINKCH field are concatenated onto the CITER value to increase the range of the CITER.

NOTE

After configuration, the TCD.CITER.E_LINK bit and the TCD.BITER.E_LINK bit must be equal or a configuration error is reported. The CITER and BITER vector widths must be equal to calculate the major loop, halfway done interrupt point.

Table 24-24 summarizes how a DMA channel can link to another DMA channel, i.e, use another channel's TCD, at the end of a loop.

Table 24-24. Channel Linking Parameters

Desired Link Behavior	TCD Control Field Name	Description
Link at end of minor loop	citer.e_link	Enable channel-to-channel linking on minor loop completion (current iteration).
	citer.linkch	Link channel number when linking at end of minor loop (current iteration).
Link at end of major loop	major.e_link	Enable channel-to-channel linking on major loop completion.
	major.linkch	Link channel number when linking at end of major loop.

24.5.8 Dynamic Programming

This section provides recommended methods to change the programming model during channel execution.

24.5.8.1 Dynamic Channel Linking and Dynamic Scatter-Gather Operation

Dynamic channel linking and dynamic scatter-gather operation is the process of changing the TCD.MAJOR.E_LINK or TCD.E_SG bits during channel execution. These bits are read from the TCD local memory at the end of channel execution thus allowing the user to enable either feature during channel execution.

Because the user is allowed to change the configuration during execution, a coherency model is needed. Consider a scenario where the user attempts to execute a dynamic channel link by enabling the TCD.MAJOR.E_LINK bit at the same time the eDMA engine is retiring the channel. The TCD.MAJOR.E_LINK would be set in the programmer's model, but it would be unclear whether the actual link was made before the channel retired.

The following coherency model is recommended when executing a dynamic channel link or dynamic scatter-gather request:

1. Set the TCD.MAJOR.E_LINK bit.
2. Read back the TCD.MAJOR.E_LINK bit
3. Test the TCD.MAJOR.E_LINK request status:
   a) If the bit is set, the dynamic link attempt was successful.
   b) If the bit is cleared, the attempted dynamic link did not succeed, the channel was already retiring.

This same coherency model is true for dynamic scatter-gather operations. For both dynamic requests, the TCD local memory controller forces the TCD.MAJOR.E_LINK and TCD.E_SG bits to zero on any writes to a channel's TCD after that channel's TCD.DONE bit is set indicating the major loop is complete.

NOTE

The user must clear the TCD.DONE bit before writing the TCD.MAJOR.E_LINK or TCD.E_SG bits. The TCD.DONE bit is cleared automatically by the eDMA engine after a channel begins execution.

Chapter 25
Fast Ethernet Controller (FEC)

25.1 Introduction

This fast ethernet control chapter of the device Reference Manual provides a feature-set overview, a functional block diagram, and transceiver connection information for both the 10 and 100 Mbps MII (media independent interface), as well as the 7-wire serial interface. Additionally, detailed descriptions of operation and the programming model are included.

NOTE

The FEC block is not implemented on the PXN20.

25.1.1 Block Diagram

The block diagram of the FEC is shown below. The FEC is implemented with a combination of hardware and microcode. The off-chip (Ethernet) interfaces are compliant with industry and IEEE® 802.3 standards.

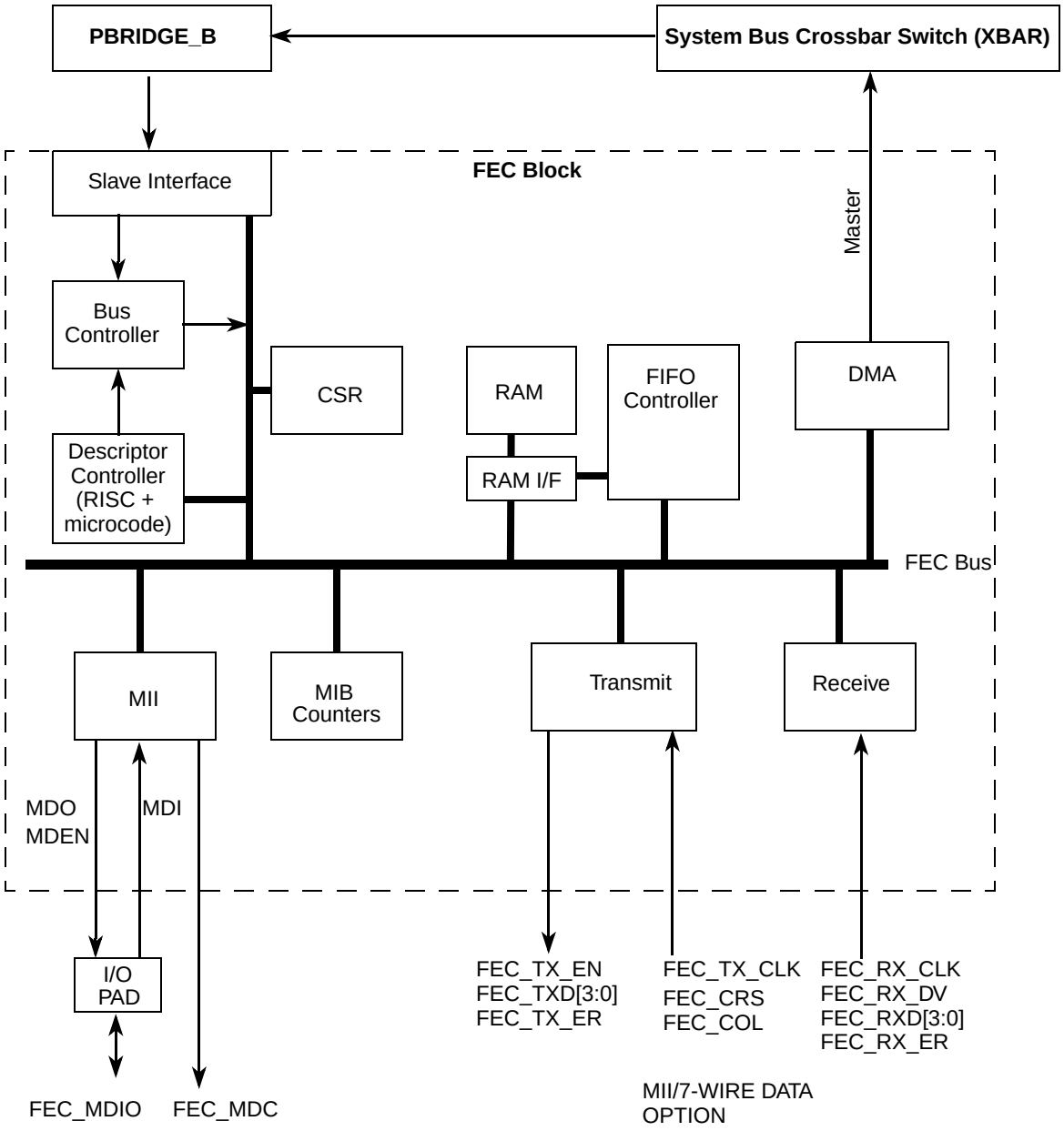


Figure 25-1. FEC Block Diagram

25.1.2 Overview

The Ethernet media access controller (MAC) is designed to support both 10 and 100 Mbps Ethernet/IEEE 802.3 networks. An external transceiver interface and transceiver function are required to complete the interface to the media. The FEC supports three different standard MAC-PHY (physical) interfaces for connection to an external Ethernet transceiver. The FEC supports the 10/100 Mbps MII and the 10 Mbps-only 7-wire interface, which uses a subset of the MII signals.

The descriptor controller is a RISC-based controller that provides the following functions in the FEC:

- Initialization (those internal registers not initialized by the user or hardware)
- High level control of the DMA channels (initiating DMA transfers)
- Interpreting buffer descriptors
- Address recognition for receive frames
- Random number generation for transmit collision backoff timer

NOTE

DMA references in this section refer to the FEC's DMA engine. This DMA engine is for the transfer of FEC data only, and is not related to the DMA controller described in Chapter 24, Enhanced Direct Memory Access Controller (eDMA).

The RAM is the focal point of all data flow in the fast Ethernet controller and is divided into transmit and receive FIFOs. The FIFO boundaries are programmable using the FRSR register. User data flows to/from the DMA block from/to the receive/transmit FIFOs. Transmit data flows from the transmit FIFO into the transmit block and receive data flows from the receive block into the receive FIFO.

The user controls the FEC by writing, through the slave interface module, into control registers located in each block. The CSR (control and status register) block provides global control (e.g., Ethernet reset and enable) and interrupt handling registers.

The MII block provides a serial channel for control/status communication with the external physical layer device (transceiver). This serial channel consists of the MDC (management data clock) and MDIO (management data input/output) lines of the MII interface.

The DMA block provides multiple channels allowing transmit data, transmit descriptor, receive data, and receive descriptor accesses to run independently.

The transmit and receive blocks provide the Ethernet MAC functionality (with some assist from microcode).

The message information block (MIB) maintains counters for a variety of network events and statistics. It is not necessary for operation of the FEC but provides valuable counters for network management. The counters supported are the RMON (RFC 1757) Ethernet Statistics group and some of the IEEE 802.3 counters.

25.1.3 Features

The FEC incorporates the following features:

- Support for three different Ethernet physical interfaces:
  - 100-Mbps IEEE 802.3 MII
  - 10-Mbps IEEE 802.3 MII
  - 10-Mbps 7-wire interface (industry standard)
- Built-in FIFO and DMA controller
- IEEE 802.3 MAC (compliant with IEEE 802.3 1998 edition)
- Programmable max frame length supports IEEE 802.1 VLAN tags and priority
- IEEE 802.3 full duplex flow control
- Support for full-duplex operation (200 Mbps throughput) with a system clock rate of 100 MHz using the external FEC_TX_CLK or FEC_RX_CLK
- Support for half-duplex operation (100 Mbps throughput) with a system clock rate of 50 MHz using the external FEC_TX_CLK or FEC_RX_CLK
- Retransmission from transmit FIFO following a collision (no system bus utilization)
- Automatic internal flushing of the receive FIFO for runts (collision fragments) and address recognition rejects (no system bus utilization)
- Address recognition
  - Frames with broadcast address may be always accepted or always rejected
  - Exact match for single 48-bit individual (unicast) address
  - Hash (64-bit hash) check of individual (unicast) addresses
  - Hash (64-bit hash) check of group (multicast) addresses
  - Promiscuous mode
- RMON and IEEE statistics
- Interrupts for network activity and error conditions

25.2 Modes of Operation

The primary operational modes are described in this section.

25.2.1 Full and Half Duplex Operation

Full duplex mode is intended for use on point-to-point links between switches or end node to switch. Half duplex mode is used in connections between an end node and a repeater or between repeaters. Selection of the duplex mode is controlled by TCR[FDEN].

When configured for full duplex mode, flow control may be enabled. Refer to the TCR[RFC_PAUSE] and TCR[TFC_PAUSE] bits, the RCR[FCE] bit, and Section 25.4.10, Full Duplex Flow Control, for more details.

Throughputs of 200 Mbps in full duplex operations and 100 Mbps in half-duplex operations can be attained.

25.2.2 Interface Options

The following interface options are supported. A detailed discussion of the interface configurations is provided in Section 25.4.5, Network Interface Options.

25.2.2.1 10 Mbps and 100 Mbps MII Interface

MII is the media-independent interface defined by the IEEE 802.3 standard for 10/100 Mbps operation. The MAC-PHY interface may be configured to operate in MII mode by asserting RCR[MII_MODE].

The speed of operation is determined by the FEC_TX_CLK and FEC_RX_CLK signals, which are driven by the external transceiver. The transceiver can auto-negotiate the speed, or it can be controlled by software via the serial management interface (FEC_MDC/FEC_MDIO signals) to the transceiver. Refer to the MMFR and MSCR register descriptions as well as the section on the MII for a description of how to read and write registers in the transceiver via this interface.

25.2.2.2 10 Mbps 7-Wire Interface Operation

The FEC supports a 7-wire interface as used by many 10 Mbps ethernet transceivers. The RCR[MII_MODE] bit controls this functionality. If this bit is deasserted, the MII mode is disabled and the 10 Mbps, 7-wire mode is enabled.

25.2.3 Address Recognition Options

The address options supported are promiscuous, broadcast reject, individual address (hash or exact match), and multicast hash match. Address recognition options are discussed in detail in Section 25.4.8, Ethernet Address Recognition.

25.2.4 Internal Loopback

Internal loopback mode is selected via RCR[LOOP]. Loopback mode is discussed in detail in Section 25.4.13, Internal and External Loopback.

25.3 Programming Model

This section gives an overview of the registers, followed by a description of the buffers.

The FEC is programmed by a combination of control/status registers (CSRs) and buffer descriptors. The CSRs are used for mode control and to extract global status information. The descriptors are used to pass data buffers and related buffer information between the hardware and software.

25.3.1 Top Level Module Memory Map

The FEC implementation requires a 1 KB memory map space. This is divided into two sections of 512 bytes each. The first is used for control/status registers. The second contains event/statistic counters held in the MIB block. Table 25-1 defines the top level memory map. All accesses to and from the FEC memory map must be via 32-bit accesses. There is no support for accesses other than 32-bit.

Table 25-1. FEC Module Memory Map

Address	Function
FFF4_C000 (Base Address) – FFF4_C1FF	Control/Status Registers
FFF4_C200 – FFF4_C3FF	MIB Block Counters

25.3.2 Detailed Memory Map (Control/Status Registers)

Table 25-2 shows the FEC register memory map with each register address, name, and a brief description. The base address of the FEC registers is 0xFFF4_C000.

NOTE

Some memory locations are not documented. The actual FEC memory map is from 0xFFF4_C000 to 0xFFF4_C5FF. Also, some bits in otherwise documented registers are not documented. These memory locations and bits are not needed for the FEC software driver. They are used mainly by the FEC subblocks for the FEC operation and happen to be visible through the slave interface.

Errant writes to these locations can corrupt FEC operation. Because the FEC is a system bus master, errant writes also can result in the corruption of any memory mapped location in the system. However, even errant writes to documented FEC memory locations can cause the same corruption.

Table 25-2. FEC Register Memory Map

Offset from FEC_BASE (0xFFF4_C000)	Register	Access[1]	Reset Value	Section/Page
0x0000–0x0003	Reserved			
0x0004	EIR—Interrupt Event Register	R/W	0x0000_0000	25.3.4.2/25-10
0x0008	EIMR—Interrupt Mask Register	R/W	0x0000_0000	25.3.4.3/25-12
0x000C–0x000F	Reserved			
0x0010	RDAR—Receive Descriptor Active Register	R/W	0x0000_0000	25.3.4.4/25-12
0x0014	TDAR—Transmit Descriptor Active Register	R/W	0x0000_0000	25.3.4.5/25-13
0x0018–0x0023	Reserved			
0x0024	ECR—Ethernet Control Register	R/W	0xF000_0000	25.3.4.6/25-14

Table 25-2. FEC Register Memory Map (continued)

Offset from FEC_BASE (0xFFF4_C000)	Register	Access[1]	Reset Value	Section/Page
0x0028–0x003F	Reserved			
0x0040	MMFR—MII Management Frame Register	R/W	U	25.3.4.7/25-15
0x0044	MSCR—MII Speed Control Register	R/W	0x0000_0000	25.3.4.8/25-16
0x0048–0x0063	Reserved			
0x0064	MIBC—MIB Control/Status Register	R/W	0xC000_0000	25.3.4.9/25-18
0x0068–0x0083	Reserved			
0x0084	RCR—Receive Control Register	R/W	0x05EE_0001	25.3.4.10/25-18
0x0088–0x00C3	Reserved			
0x00C4	TCR—Transmit Control Register	R/W	0x0000_0000	25.3.4.11/25-20
0x00C8–0x00E3	Reserved			
0x00E4	PALR—MAC Address Low Register	R/W	U	25.3.4.12/25-21
0x00E8	PAUR—MAC Address Upper Register + Type Field	R/W	0xUUUU_8808	25.3.4.13/25-21
0x00EC	OPD—Opcode + Pause Duration Fields	R/W	0x0001_UUUU	25.3.4.14/25-22
0x00F0–0x0117	Reserved			
0x0118	IAUR—Upper 32 bits of Individual Hash Table	R/W	U	25.3.4.15/25-23
0x011C	IALR—Lower 32 Bits of Individual Hash Table	R/W	U	25.3.4.16/25-23
0x0120	GAUR—Upper 32 bits of Group Hash Table	R/W	U	25.3.4.17/25-24
0x0124	GALR—Lower 32 bits of Group Hash Table	R/W	U	25.3.4.18/25-25
0x0128–0x0143	Reserved			
0x0144	TFWR—Transmit FIFO Watermark	R/W	0x0000_0000	25.3.4.19/25-25
0x0148–0x014B	Reserved			
0x014C	FRBR—FIFO Receive Bound Register	R/W	0x0000_0600	25.3.4.20/25-26
0x0150	FRSR—FIFO Receive FIFO Start Registers	R/W	0x0000_0500	25.3.4.21/25-27
0x0154–0x017F	Reserved			
0x0180	ERDSR—Pointer to Receive Descriptor Ring	R/W	U	25.3.4.22/25-27
0x0184	ETDSR—Transmit Buffer Descriptor Ring Start Register	R/W	U	25.3.4.23/25-28
0x0188	EMRBR—Receive Buffer Size Register	R/W	U	25.3.4.24/25-29
0x018C–0x3FFF	Reserved			

1 All accesses to and from the FEC memory map must be via 32-bit accesses. There is no support for accesses other than 32-bit.

25.3.3 MIB Block Counters Memory Map

Table 25-3 defines the MIB Counters memory map which defines the locations in the MIB RAM space where hardware-maintained counters reside. These fall in the 0xFFF4_C200 – 0xFFF4_C3FF address offset range. The counters are divided into two groups.

- RMON counters are included which cover the Ethernet Statistics counters defined in RFC 1757. In addition to the counters defined in the Ethernet Statistics group, a counter is included to count truncated frames as the FEC only supports frame lengths up to a maximum of 2047 bytes. The RMON counters are implemented independently for transmit and receive to ensure accurate network statistics when operating in full duplex mode.
- IEEE counters are included which support the Mandatory and Recommended counter packages defined in section 5 of ANSI/IEEE Std. 802.3 (1998 edition). The IEEE Basic Package objects are supported by the FEC but do not require counters in the MIB block. In addition, some of the recommended package objects which are supported do not require MIB counters. Counters for transmit and receive full duplex flow control frames are included as well.

Table 25-3. MIB Counters Memory Map

Offset from FEC_BASE (0xFFF4_C000)[1]	Mnemonic	Description
0x0200	RMON_T_DROP	Count of frames not counted correctly
0x0204	RMON_T_PACKETS	RMON Tx packet count
0x0208	RMON_T_BC_PKT	RMON Tx Broadcast Packets
0x020C	RMON_T_MC_PKT	RMON Tx Multicast Packets
0x0210	RMON_T_CRC_ALIGN	RMON Tx Packets w CRC/Align error
0x0214	RMON_T_UNDERSIZE	RMON Tx Packets < 64 bytes, good crc
0x0218	RMON_T_OVERSIZE	RMON Tx Packets > MAX_FL bytes, good crc
0x021C	RMON_T_FRAG	RMON Tx Packets < 64 bytes, bad crc
0x0220	RMON_T_JAB	RMON Tx Packets > MAX_FL bytes, bad crc
0x0224	RMON_T_COL	RMON Tx collision count
0x0228	RMON_T_P64	RMON Tx 64 byte packets
0x022C	RMON_T_P65TO127	RMON Tx 65 to 127 byte packets
0x0230	RMON_T_P128TO255	RMON Tx 128 to 255 byte packets
0x0234	RMON_T_P256TO511	RMON Tx 256 to 511 byte packets
0x0238	RMON_T_P512TO1023	RMON Tx 512 to 1023 byte packets
0x023C	RMON_T_P1024TO2047	RMON Tx 1024 to 2047 byte packets
0x0240	RMON_T_P_GTE2048	RMON Tx packets w > 2048 bytes
0x0244	RMON_T_OCTETS	RMON Tx Octets
0x0248	IEEE_T_DROP	Count of frames not counted correctly

Table 25-3. MIB Counters Memory Map (continued)

Offset from FEC_BASE (0xFFF4_C000)[1]	Mnemonic	Description
0x024C	IEEE_T_FRAME_OK	Frames Transmitted OK
0x0250	IEEE_T_1COL	Frames Transmitted with Single Collision
0x0254	IEEE_T_MCOL	Frames Transmitted with Multiple Collisions
0x0258	IEEE_T_DEF	Frames Transmitted after Deferral Delay
0x025C	IEEE_T_LCOL	Frames Transmitted with Late Collision
0x0260	IEEE_T_EXCOL	Frames Transmitted with Excessive Collisions
0x0264	IEEE_T_MACERR	Frames Transmitted with Tx FIFO Underrun
0x0268	IEEE_T_CSERR	Frames Transmitted with Carrier Sense Error
0x026C	IEEE_T_SQE	Frames Transmitted with SQE Error
0x0270	IEEE_T_FDXFC	Flow Control Pause frames transmitted
0x0274	IEEE_T_OCTETS_OK	Octet count for Frames Transmitted w/o Error
0x0280	RMON_R_DROP	Count of frames not counted correctly
0x0284	RMON_R_PACKETS	RMON Rx packet count
0x0288	RMON_R_BC_PKT	RMON Rx Broadcast Packets
0x028C	RMON_R_MC_PKT	RMON Rx Multicast Packets
0x0290	RMON_R_CRC_ALIGN	RMON Rx Packets w CRC/Align error
0x0294	RMON_R_UNDERSIZE	RMON Rx Packets < 64 bytes, good crc
0x0298	RMON_R_OVERSIZE	RMON Rx Packets > MAX_FL bytes, good crc
0x029C	RMON_R_FRAG	RMON Rx Packets < 64 bytes, bad crc
0x02A0	RMON_R_JAB	RMON Rx Packets > MAX_FL bytes, bad crc
0x02A4	—	Reserved
0x02A8	RMON_R_P64	RMON Rx 64 byte packets
0x02AC	RMON_R_P65TO127	RMON Rx 65 to 127 byte packets
0x02B0	RMON_R_P128TO255	RMON Rx 128 to 255 byte packets
0x02B4	RMON_R_P256TO511	RMON Rx 256 to 511 byte packets
0x02B8	RMON_R_P512TO1023	RMON Rx 512 to 1023 byte packets
0x02BC	RMON_R_P1024TO2047	RMON Rx 1024 to 2047 byte packets
0x02C0	RMON_R_P_GTE2048	RMON Rx packets w > 2048 bytes
0x02C4	RMON_R_OCTETS	RMON Rx Octets
0x02C8	IEEE_R_DROP	Count of frames not counted correctly
0x02CC	IEEE_R_FRAME_OK	Frames Received OK

Table 25-3. MIB Counters Memory Map (continued)

Offset from FEC_BASE (0xFFF4_C000)[1]	Mnemonic	Description
0x02D0	IEEE_R_CRC	Frames Received with CRC Error
0x02D4	IEEE_R_ALIGN	Frames Received with Alignment Error
0x02D8	IEEE_R_MACERR	Receive Fifo Overflow count
0x02DC	IEEE_R_FDXFC	Flow Control Pause frames received
0x02E0	IEEE_R_OCTETS_OK	Octet count for Frames Rcvd w/o Error

1 All accesses to and from the FEC memory map must be via 32-bit accesses. There is no support for accesses other than 32-bit.

25.3.4 Registers

25.3.4.1 FEC Burst Optimization Master Control Register (FBOMCR)

Although *not* an FEC register, the FEC burst optimization master control register (FBOMCR) controls FEC burst optimization behavior on the system bus, hence it is mentioned here. Full details are provided in Section 19.2.2.1, FEC Burst Optimization Master Control Register (FBOMCR). FEC registers are described in Section 25.3.4.21, FIFO Receive Start Register (FRSR), through Section 25.3.4.24, Receive Buffer Size Register (EMRBR).

In order to increase throughput, the FEC interface to the system bus can accumulate read requests or writes to burst those transfers on the system bus. The FBOMCR determines the XBAR ports for which this bursting is enabled, as well as whether the bursting is for reads, writes, or both. FBOMCR also controls how errors for writes are handled. The FBOMCR address is 0xFFF4_0024, which is the ECSM base address 0xFFF4_0000 plus the offset of 0x0024.

25.3.4.2 Ethernet Interrupt Event Register (EIR)

When an event occurs that sets a bit in the EIR, an interrupt is generated if the corresponding bit in the interrupt mask register (EIMR) is also set. The bit in the EIR is cleared if a one is written to that bit position; writing zero has no effect. This register is cleared on hardware reset.

These interrupts can be divided into operational interrupts, transceiver/network error interrupts, and internal error interrupts. Interrupts which may occur in normal operation are GRA, TXF, TXB, RXF, RXB, and MII. Interrupts resulting from errors/problems detected in the network or transceiver are HBERR, BABR, BABT, LC, and RL. Interrupts resulting from internal errors are HBERR and UN.

Some of the error interrupts are independently counted in the MIB block counters. Software may choose to mask off these interrupts, since these errors are visible to network management via the MIB counters:

- HBERR – IEEE_T_SQE
- BABR – RMON_R_OVERSIZE (good CRC), RMON_R_JAB (bad CRC)
- BABT – RMON_T_OVERSIZE (good CRC), RMON_T_JAB (bad CRC)

- LATE_COL – IEEE_T_LCOL
- COL_RETRY_LIM – IEEE_T_EXCOL
- XFIFO_UN – IEEET_MACERR

Offset: FEC_BASE + 0x0004 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	HB ERR	BABR	BABT	GRA	TXF	TXB	RXF	RXB	MII	EB ERR	LC	RL	UN	0	0	0
W	w1c[1]	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 25-2. Ethernet Interrupt Event Register (EIR)

1 "w1c" signifies the bit is cleared by writing 1 to it.

Table 25-4. EIR Field Descriptions

Field	Description
HBERR	Heartbeat error. This interrupt indicates that HBC is set in the TCR register and that the COL input was not asserted within the Heartbeat window following a transmission.
BABR	Babbling receive error. This bit indicates a frame was received with length in excess of RCR[MAX_FL] bytes.
BABT	Babbling transmit error. This bit indicates that the transmitted frame length has exceeded RCR[MAX_FL] bytes. This condition is usually caused by a frame that is too long being placed into the transmit data buffers. Truncation does not occur.
GRA	Graceful stop complete. This interrupt is asserted for one of three reasons. Graceful stop means that the transmitter is put into a pause state after completion of the frame currently being transmitted. • A graceful stop, which was initiated by the setting of the TCR[GTS] bit is now complete. • A graceful stop, which was initiated by the setting of the TCR[TFC_PAUSE] bit is now complete. • A graceful stop, which was initiated by the reception of a valid full duplex flow control "pause" frame is now complete.
TXF	Transmit frame interrupt. This bit indicates that a frame has been transmitted and that the last corresponding buffer descriptor has been updated.
TXB	Transmit buffer interrupt. This bit indicates that a transmit buffer descriptor has been updated.
RXF	Receive frame interrupt. This bit indicates that a frame has been received and that the last corresponding buffer descriptor has been updated.
RXB	Receive buffer interrupt. This bit indicates that a receive buffer descriptor has been updated that was not the last in the frame.
MII	MII interrupt. This bit indicates that the MII has completed the data transfer requested.
EBERR	Ethernet bus error. This bit indicates that a system bus error occurred when a DMA transaction was underway. When the EBERR bit is set, ECR[ETHER_EN] is cleared, halting frame processing by the FEC. When this occurs, software must ensure that the FIFO controller and DMA are also soft reset.

Table 25-4. EIR Field Descriptions (continued)

Field	Description
LC	Late collision. This bit indicates that a collision occurred beyond the collision window (slot time) in half duplex mode. The frame is truncated with a bad CRC and the remainder of the frame is discarded.
RL	Collision retry limit. This bit indicates that a collision occurred on each of 16 successive attempts to transmit the frame. The frame is discarded without being transmitted, and transmission of the next frame begins. Can only occur in half duplex mode.
UN	Transmit FIFO underrun. This bit indicates that the transmit FIFO became empty before the complete frame was transmitted. A bad CRC is appended to the frame fragment and the remainder of the frame is discarded.

25.3.4.3 Ethernet Interrupt Mask Register (EIMR)

The EIMR register controls which interrupt events are allowed to generate actual interrupts. All implemented bits in this CSR are read/write. This register is cleared on a hardware reset. If the corresponding bits in both the EIR and EIMR registers are set, the interrupt is signalled to the CPU. The interrupt signal remains asserted until a 1 is written to the EIR bit (write 1 to clear) or a 0 is written to the EIMR bit.

Offset: FEC_BASE + 0x0008 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	HB ERR	BABR	BABT	GRA	TXF	TXB	RXF	RXB	MII	EB ERR	LC	RL	UN	0	0	0
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 25-3. Interrupt Mask Register (EIMR)

Table 25-5. EIMR Field Descriptions

Field	Description
See Figure 25-2 and Table 25-4	Interrupt mask. Each bit corresponds to an interrupt source defined by the EIR register. The corresponding EIMR bit determines whether an interrupt condition can generate an interrupt. 0 The corresponding interrupt source is masked. 1 The corresponding interrupt source is not masked. Write 1 to clear.

25.3.4.4 Receive Descriptor Active Register (RDAR)

RDAR is a command register, written by the user, that indicates that the receive descriptor ring has been updated (empty receive buffers have been produced by the driver with the empty bit set).

Whenever the register is written, the R_DES_ACTIVE bit is set. This is independent of the data actually written by the user. When set, the FEC polls the receive descriptor ring and processes receive frames

(provided ECR[ETHER_EN] is also set). Once the FEC polls a receive descriptor whose empty bit is not set, the FEC clears R_DES_ACTIVE and ceases receive descriptor ring polling until the user sets the bit again, signifying that additional descriptors have been placed into the receive descriptor ring.

The RDAR register is cleared at reset and when ECR[ETHER_EN] is cleared.

Offset: FEC_BASE + 0x0010 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	R_DES_ACTIVE	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 25-4. Receive Descriptor Active Register (RDAR)

Table 25-6. RDAR Field Descriptions

Field	Description
0–6	Reserved, should be cleared.
R_DES_ACTIVE	Set to one when this register is written, regardless of the value written. Cleared by the FEC device whenever no additional “empty” descriptors remain in the receive ring. Also cleared when ECR[ETHER_EN] is cleared.
8–31	Reserved, should be cleared.

25.3.4.5 Transmit Descriptor Active Register (TDAR)

The TDAR is a command register that should be written by the user to indicate that the transmit descriptor ring has been updated (transmit buffers have been produced by the driver with the ready bit set in the buffer descriptor).

Whenever the register is written, the X_DES_ACTIVE bit is set. This value is independent of the data actually written by the user. When set, the FEC polls the transmit descriptor ring and processes transmit frames (provided ECR[ETHER_EN] is also set). Once the FEC polls a transmit descriptor whose ready bit is not set, the FEC clears X_DES_ACTIVE and ceases transmit descriptor ring polling until the user sets the bit again, signifying additional descriptors have been placed into the transmit descriptor ring.

The TDAR register is cleared at reset, when ECR[ETHER_EN] is cleared, or when ECR[RESET] is set.

Offset: FEC_BASE + 0x0014 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	X_DES_ACTIVE	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 25-5. Transmit Descriptor Active Register (TDAR)

Table 25-7. TDAR Field Descriptions

Field	Description
0–6	Reserved, should be cleared.
X_DES_ACTIVE	Set to one when this register is written, regardless of the value written. Cleared by the FEC device whenever no additional “ready” descriptors remain in the transmit ring. Also cleared when ECR[ETHER_EN] is cleared.
8–31	Reserved, should be cleared.

25.3.4.6 Ethernet Control Register (ECR)

ECR is a read/write user register, though both fields in this register may be altered by hardware as well. The ECR is used to enable/disable the FEC.

Offset: FEC_BASE + 0x0024 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	ETHER_EN	RESET
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 25-6. Ethernet Control Register (ECR)

Table 25-8. ECR Field Descriptions

Bits	Description
ETHER_EN	When this bit is set, the FEC is enabled, and reception and transmission are possible. When this bit is cleared, reception is immediately stopped and transmission is stopped after a bad CRC is appended to any currently transmitted frame. The buffer descriptors for an aborted transmit frame are not updated after clearing this bit. When ETHER_EN is deasserted, the DMA, buffer descriptor, and FIFO control logic are reset, including the buffer descriptor and FIFO pointers. The ETHER_EN bit is altered by hardware under the following conditions: • ECR[RESET] is set by software, in which case ETHER_EN is cleared • An error condition causes the EIR[EBERR] bit to set, in which case ETHER_EN is cleared
RESET	When this bit is set, the equivalent of a hardware reset is performed but it is local to the FEC. ETHER_EN is cleared and all other FEC registers take their reset values. Also, any transmission/reception currently in progress is abruptly aborted. This bit is automatically cleared by hardware during the reset sequence. The reset sequence takes approximately 8 system clock cycles after RESET is written with a 1.

25.3.4.7 MII Management Frame Register (MMFR)

The MMFR is accessed by the user and does not reset to a defined value. The MMFR register is used to communicate with the attached MII compatible PHY devices, providing read/write access to their MII registers. Performing a write to the MMFR causes a management frame to be sourced unless the MSCR has been programmed to 0. In the case of writing to MMFR when MSCR = 0, if the MSCR register is then written to a non-zero value, an MII frame is generated with the data previously written to the MMFR. This allows MMFR and MSCR to be programmed in either order if MSCR is currently zero.

Offset: FEC_BASE + 0x0040 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	ST				OP				PA				RA			TA
W																
Reset	U[1]	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	DATA															
W																
Reset	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U

Figure 25-7. MII Management Frame Register (MMFR)

1 “U” signifies a bit that is uninitialized.

Table 25-9. MMFR Field Descriptions

Field	Description
ST	Start of frame delimiter. These bits must be programmed to 01 for a valid MII management frame.
OP	Operation code. This field must be programmed to 10 (read) or 01 (write) to generate a valid MII management frame. A value of 11 produces “read” frame operation. A value of 00 produces “write” frame operation, but these frames are not MII compliant.
PA	PHY address. This field specifies one of as many as 32 attached PHY devices.
RA	Register address. This field specifies one of as many as 32 registers within the specified PHY device.
TA	Turn around. This field must be programmed to 10 to generate a valid MII management frame.
DATA	Management frame data. This is the field for data to be written to or read from the PHY register.

To perform a read or write operation on the MII management interface, the MMFR register must be written by the user. To generate a valid read or write management frame, the ST field must be written with a 01 pattern, and the TA field must be written with a 10. If other patterns are written to these fields, a frame is generated that does not comply with the IEEE 802.3 MII definition.

To generate an IEEE 802.3-compliant MII management interface write frame (write to a PHY register), the user must write {01 01 PHYAD REGAD 10 DATA} to the MMFR register. Writing this pattern causes the control logic to shift out the data in the MMFR register following a preamble generated by the control state machine. During this time, the contents of the MMFR register are altered as the contents are serially shifted and are unpredictable if read by the user. Once the write management frame operation has completed, the MII interrupt is generated. At this time the contents of the MMFR register match the original value written.

To generate an MII management interface read frame (read a PHY register) the user must write {01 10 PHYAD REGAD 10 XXXX} to the MMFR register (the content of the DATA field is a “don’t care”). Writing this pattern causes the control logic to shift out the data in the MMFR register following a preamble generated by the control state machine. During this time, the contents of the MMFR register are altered as the contents are serially shifted, and are unpredictable if read by the user. Once the read management frame operation has completed, the MII interrupt is generated. At this time, the contents of the MMFR register match the original value written, except for the DATA field, whose contents have been replaced by the value read from the PHY register.

If the MMFR register is written while frame generation is in progress, the frame contents are altered. Software should software should poll the EIR[MII] bit or use the EIR[MII] bit to generate an interrupt to avoid writing to the MMFR register while frame generation is in progress.

25.3.4.8 MII Speed Control Register (MSCR)

The MSCR provides control of the MII clock (FEC_MDC signal) frequency, allows a preamble drop on the MII management frame, and provides observability (intended for manufacturing test) of an internal counter used in generating the FEC_MDC clock signal.

Offset: FEC_BASE + 0x0044 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	DIS_PREAMBLE	MII_SPEED						0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 25-8. MII Speed Control Register (MSCR)

Table 25-10. MSCR Field Descriptions

Field	Description
0–23	Reserved, should be cleared.
DIS_PREAMBLE	Asserting this bit causes preamble (32 1's) not to be prepended to the MII management frame. The MII standard allows the preamble to be dropped if the attached PHY devices does not require it.
MII_SPEED	MII_SPEED controls the frequency of the MII management interface clock (FEC_MDC) relative to the system clock. A value of 0 in this field "turns off" the MDC and leaves it in low voltage state. Any non-zero value results in the MDC frequency of 1/(MII_SPEED * 4) of the system clock frequency.
31	Reserved, should be cleared.

The MII_SPEED field must be programmed with a value to provide an MDC frequency of less than or equal to 2.5 MHz to be compliant with the IEEE 802.3 MII specification. The MII_SPEED must be set to a non-zero value in order to source a read or write management frame. After the management frame is complete the MSCR register may optionally be set to zero to turn off the MDC. The MDC generated has a 50% duty cycle except when MII_SPEED is changed during operation (change takes effect following either a rising or falling edge of MDC).

If the system clock is 50 MHz, programming this register to 0x0000_0005 results in an MDC frequency of 50 MHz × 1/20 = 2.5 MHz. A table showing optimum values for MII_SPEED as a function of system clock frequency is provided in Table 25-11.

Table 25-11. Programming Examples for MSCR

System Clock Frequency	MII_SPEED (field in reg)	MDC frequency
50 MHz	0x5	2.5 MHz
66 MHz	0x7	2.36 MHz
80 MHz	0x8	2.5 MHz
100 MHz	0xA	2.5 MHz
132 MHz[1]	0xD	2.5 MHz

[1] **Note:** Observe maximum system clock frequency when programming MII_SPEED.

25.3.4.9 MIB Control Register (MIBC)

The MIBC is a read/write register used to provide control of and to observe the state of the MIB block. This register is accessed by user software if there is a need to disable the MIB block operation. For example, in order to clear all MIB counters in RAM the user should disable the MIB block, then clear all the MIB RAM locations, then enable the MIB block. The MIB_DISABLE bit is reset to 1. See Table 25-3 for the locations of the MIB counters.

Offset: FEC_BASE + 0x0064 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	MIB_DISABLE	MIB_IDLE	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 25-9. MIB Control Register (MIBC)

Table 25-12. MIBC Field Descriptions

Field	Description
MIB_DISABLE	A read/write control bit. If set, the MIB logic halts and does not update any MIB counters.
MIB_IDLE	A read-only status bit. If set, the MIB block is not currently updating any MIB counters.

25.3.4.10 Receive Control Register (RCR)

The RCR is programmed by the user. The RCR controls the operational mode of the receive block and

should be written only when ECR[ETHER_EN] = 0 (initialization time).

Offset: FEC_BASE + 0x0084 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	MAX_FL										
W																
Reset	0	0	0	0	0	1	0	1	1	1	1	0	1	1	1	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	FCE	BC_REJ	PROM	MII_MODE	DRT	LOOP
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 25-10. Receive Control Register (RCR)

Table 25-13. RCR Field Descriptions

Field	Description
0–4	Reserved, should be cleared.
MAX_FL	Maximum frame length. Resets to decimal 1518. Length is measured starting at DA and includes the CRC at the end of the frame. Transmit frames longer than MAX_FL cause the BABT interrupt to occur. Receive frames longer than MAX_FL cause a BABR interrupt and sets the LG bit in the end-of-frame receive buffer descriptor. You can program the default value to 1518 or 1522 (if VLAN tags are supported).
16–25	Reserved, should be cleared.
FCE	Flow control enable. If asserted, the receiver detects PAUSE frames. Upon PAUSE frame detection, the transmitter stops transmitting data frames for a given duration.
BC_REJ	Broadcast frame reject. If asserted, frames with DA (destination address) = FF_FF_FF_FF_FF_FF is rejected unless the PROM bit is set. If both BC_REJ and PROM = 1, then frames with broadcast DA is accepted and the M (MISS) bit is set in the receive buffer descriptor.
PROM	Promiscuous mode. All frames are accepted regardless of address matching.
MII_MODE	Media independent interface mode. Selects external interface mode. Setting this bit to one selects MII mode, setting this bit equal to zero selects 7-wire mode (used only for serial 10 Mbps). This bit controls the interface mode for both transmit and receive blocks.
DRT	Disable receive on transmit. 0 Receive path operates independently of transmit (use for full duplex or to monitor transmit activity in half duplex mode). 1 Disable reception of frames while transmitting (normally used for half duplex mode).
LOOP	Internal loopback. If set, transmitted frames are looped back internal to the device and the transmit output signals are not asserted. The system clock is substituted for the FEC_TX_CLK when LOOP is asserted. DRT must be set to zero when asserting LOOP.

25.3.4.11 Transmit Control Register (TCR)

The TCR is read/write and is written by the user to configure the transmit block. This register is cleared at system reset. Bits 29 and 30 should be modified only when ECR[ETHER_EN] = 0.

Offset: FEC_BASE + 0x00C4 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	RFC_ PAUSE	TFC_ PAUSE	FDEN	HBC	GTS
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 25-11. Transmit Control Register (TCR)

Table 25-14. TCR Field Descriptions

Field	Description
0–26	Reserved, should be cleared.
RFC_PAUSE	Receive frame control pause. This read-only status bit is asserted when a full duplex flow control pause frame has been received and the transmitter is paused for the duration defined in this pause frame. This bit is automatically cleared when the pause duration is complete.
TFC_PAUSE	Transmit frame control pause. Transmits a PAUSE frame when asserted. When this bit is set, the MAC stops transmitting data frames after the current transmission is complete. At this time, the GRA interrupt in the EIR register is asserted. With transmission of data frames stopped, the MAC transmits a MAC Control PAUSE frame. Next, the MAC clears the TFC_PAUSE bit and resumes transmitting data frames. Note that if the transmitter is paused due to user assertion of GTS or reception of a PAUSE frame, the MAC may still transmit a MAC Control PAUSE frame.
FDEN	Full duplex enable. If set, frames are transmitted independent of carrier sense and collision inputs. This bit should only be modified when ETHER_EN is deasserted.
HBC	Heartbeat control. If set, the heartbeat check is performed following end of transmission and the HB bit in the status register is set if the collision input does not assert within the heartbeat window. This bit should only be modified when ETHER_EN is deasserted.
GTS	Graceful transmit stop. When this bit is set, the MAC stops transmission after any frame that is currently being transmitted is complete and the GRA interrupt in the EIR register is asserted. If frame transmission is not currently underway, the GRA interrupt is asserted immediately. Once transmission has completed, a “restart” can be accomplished by clearing the GTS bit. The next frame in the transmit FIFO is then transmitted. If an early collision occurs during transmission when GTS = 1, transmission stops after the collision. The frame is transmitted again once GTS is cleared. Note that there may be old frames in the transmit FIFO that are transmitted when GTS is reasserted. To avoid this deassert ECR[ETHER_EN] following the GRA interrupt.

25.3.4.12 Physical Address Low Register (PALR)

The PALR is written by the user. This register contains the lower 32 bits (bytes 0,1,2,3) of the 48-bit MAC address used in the address recognition process to compare with the DA (destination address) field of receive frames with an individual DA. In addition, this register is used in bytes 0 through 3 of the 6-byte source address field when transmitting PAUSE frames. This register is not reset and must be initialized by the user.

Offset: FEC_BASE + 0x00E4 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	PADDR1															
Reset	U[1]	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	PADDR1															
Reset	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U

Figure 25-12. Physical Address Low Register (PALR)

1 "U" signifies a bit that is uninitialized. Refer to the Preface of the book.

Table 25-15. PALR Field Descriptions

Field	Description
PADDR1	Bytes 0 (bits 0:7), 1 (bits 8:15), 2 (bits 16:23) and 3 (bits 24:31) of the 6-byte individual address to be used for exact match, and the Source Address field in PAUSE frames.

25.3.4.13 Physical Address Upper Register (PAUR)

The PAUR is written by the user. This register contains the upper 16 bits (bytes 4 and 5) of the 48-bit MAC address used in the address recognition process to compare with the DA (destination address) field of receive frames with an individual DA. In addition, this register is used in bytes 4 and 5 of the 6-byte Source Address field when transmitting PAUSE frames. Bits 16:31 of PAUR contain a constant TYPE field (0x8808) used for transmission of PAUSE frames.This register is not reset, and bits 0:15 must be initialized by the user. Refer to Section 25.4.10, Full Duplex Flow Control, for information on using the TYPE field.

Offset: FEC_BASE + 0x00E8 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	PADDR2															
W																
Reset	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	TYPE															
W																
Reset	1	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0

Figure 25-13. Physical Address Upper Register (PAUR)

Table 25-16. PAUR Field Descriptions

Field	Description
PADDR2	Bytes 4 (bits 0:7) and 5 (bits 8:15) of the 6-byte individual address to be used for exact match, and the Source Address field in PAUSE frames.
TYPE	The type field is used in PAUSE frames. These bits are a constant, 0x8808.

25.3.4.14 Opcode/Pause Duration Register (OPD)

The OPD is read/write accessible. This register contains the 16-bit OPCODE and 16-bit pause duration (PAUSE_DUR) fields used in transmission of a PAUSE frame. The OPCODE field is a constant value, 0x0001. When another node detects a PAUSE frame, that node pauses transmission for the duration specified in the pause duration field. This register is not reset and must be initialized by the user. Refer to Section 25.4.10, Full Duplex Flow Control, for information on using the OPD register.

Offset: FEC_BASE + 0x00EC Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	OPCODE															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	PAUSE_DUR															
W																
Reset	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U

Figure 25-14. Opcode/Pause Duration Register (OPD)

Table 25-17. OPD Field Descriptions

Field	Description
OPCODE	Opcode field used in PAUSE frames. These bits are a constant, 0x0001.
PAUSE_DUR	Pause duration field used in PAUSE frames.

25.3.4.15 Descriptor Individual Upper Address Register (IAUR)

The IAUR is written by the user. This register contains the upper 32 bits of the 64-bit individual address hash table used in the address recognition process to check for possible match with the DA field of receive frames with an individual DA. This register is not reset and must be initialized by the user.

Offset: FEC_BASE + 0x0118 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	IADDR1															
Reset	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	IADDR1															
Reset	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U

Figure 25-16. Descriptor Individual Upper Address Register (IAUR)

Table 25-18. IAUR Field Descriptions

Field	Descriptions
IADDR1	The upper 32 bits of the 64-bit hash table used in the address recognition process for receive frames with a unicast address. Bit 31 of IADDR1 contains hash index bit 63. Bit 0 of IADDR1 contains hash index bit 32.

25.3.4.16 Descriptor Individual Lower Address (IALR)

The IALR register is written by the user. This register contains the lower 32 bits of the 64-bit individual address hash table used in the address recognition process to check for possible match with the DA field of receive frames with an individual DA. This register is not reset and must be initialized by the user.

Offset: FEC_BASE + 0x011C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	IADDR2															
Reset	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	IADDR2															
Reset	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U

Figure 25-17. Descriptor Individual Lower Address (IALR)

Table 25-19. IALR Field Descriptions

Field	Description
IADDR2	The lower 32 bits of the 64-bit hash table used in the address recognition process for receive frames with a unicast address. Bit 31 of IADDR2 contains hash index bit 31. Bit 0 of IADDR2 contains hash index bit 0.

25.3.4.17 Descriptor Group Upper Address (GAUR)

The GAUR is written by the user. This register contains the upper 32 bits of the 64-bit hash table used in the address recognition process for receive frames with a multicast address. This register must be initialized by the user.

Offset: FEC_BASE + 0x0120 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	GADDR1															
Reset	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	GADDR1															
Reset	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U

Figure 25-18. Descriptor Group Upper Address Register (GAUR)

Table 25-20. GAUR Field Descriptions

Field	Description
GADDR1	The GADDR1 register contains the upper 32 bits of the 64-bit hash table used in the address recognition process for receive frames with a multicast address. Bit 31 of GADDR1 contains hash index bit 63. Bit 0 of GADDR1 contains hash index bit 32.

25.3.4.18 Descriptor Group Lower Address (GALR)

The GALR register is written by the user. This register contains the lower 32 bits of the 64-bit hash table used in the address recognition process for receive frames with a multicast address. This register must be initialized by the user.

Offset: FEC_BASE + 0x0124 Access: User read/write

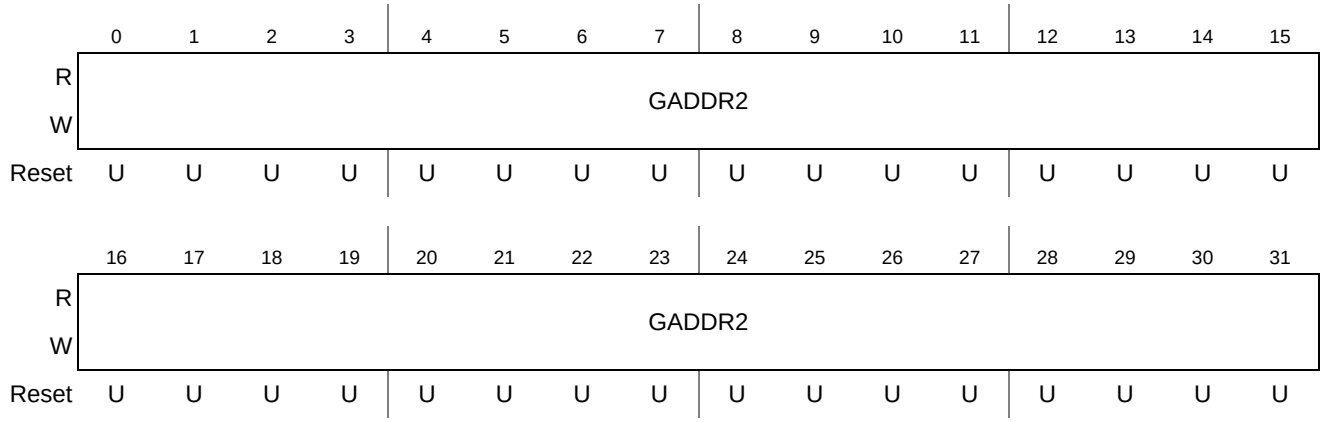


Figure 25-19. Descriptor Group Lower Address Register (GALR)

Table 25-21. GALR Field Descriptions

Field	Description
GADDR2	The GADDR2 register contains the lower 32 bits of the 64-bit hash table used in the address recognition process for receive frames with a multicast address. Bit 31 of GADDR2 contains hash index bit 31. Bit 0 of GADDR2 contains hash index bit 0.

25.3.4.19 FIFO Transmit FIFO Watermark Register (TFWR)

The TFWR is a 32-bit read/write register with one 2-bit field programmed by the user to control the amount of data required in the transmit FIFO before transmission of a frame can begin. This allows the user to minimize transmit latency (TFWR = 0x) or allow for larger bus access latency (TFWR = 11) due to contention for the system bus. Setting the watermark to a high value minimizes the risk of transmit FIFO underrun due to contention for the system bus. The byte counts associated with the TFWR field may need to be modified to match a given system requirement (worst case bus access latency by the transmit data DMA channel).

Offset: FEC_BASE + 0x0144 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	X_WMRK	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 25-20. FIFO Transmit FIFO Watermark Register (TFWR)

Table 25-22. TFWR Field Descriptions

Field	Descriptions
0–29	Reserved, should be cleared.
X_WMRK	Number of bytes written to transmit FIFO before transmission of a frame begins 0x 64 bytes written 10 128 bytes written 11 192 bytes written

25.3.4.20 FIFO Receive Bound Register (FRBR)

The FRBR is a 32-bit register with one 8-bit field that the user can read to determine the upper address bound of the FIFO RAM. Drivers can use this value, along with the FRSR register, to appropriately divide the available FIFO RAM between the transmit and receive data paths.

Offset: FEC_BASE + 0x014C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	R_BOUND								0	0
W																
Reset	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0

Figure 25-21. FIFO Receive Bound Register (FRBR)

Table 25-23. FRBR Field Descriptions

Field	Descriptions
0–21	Reserved, read as 0 (except bit 21, which is read as 1).
R_BOUND	Read-only. Highest valid FIFO RAM address.
30–31	Reserved, should be cleared.

25.3.4.21 FIFO Receive Start Register (FRSR)

The FRSR is a 32-bit register with one 8-bit field programmed by the user to indicate the starting address of the receive FIFO. FRSR marks the boundary between the transmit and receive FIFOs. The transmit FIFO uses addresses from the start of the FIFO to the location four bytes before the address programmed into the FRSR. The receive FIFO uses addresses from FRSR to FRBR inclusive.

The FRSR register is initialized by hardware at reset. FRSR only needs to be written to change the default value.

Offset: FEC_BASE + 0x0150 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	R_FSTART								0	0
W																
Reset	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0

Figure 25-22. FIFO Receive Start Register (FRSR)

Table 25-24. FRSR Field Descriptions

Field	Descriptions
0–21	Reserved, read as 0 (except bit 21, which is read as 1).
R_FSTART	Address of first receive FIFO location. Acts as delimiter between receive and transmit FIFOs.
30–31	Reserved, read as 0.

25.3.4.22 Receive Descriptor Ring Start (ERDSR)

The ERDSR is written by the user. It provides a pointer to the start of the circular receive buffer descriptor queue in external memory. This pointer must be 32-bit aligned; however, it is recommended it be made 128-bit aligned (evenly divisible by 16).

This register is not reset and must be initialized by the user prior to operation.

Offset: FEC_BASE + 0x0180 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	R_DES_START															
Reset	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	R_DES_START														0	0
Reset	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U

Figure 25-23. Receive Descriptor Ring Start Register (ERDSR)

Table 25-25. ERDSR Field Descriptions

Field	Descriptions
R_DES_START	Pointer to start of receive buffer descriptor queue.
30–31	Reserved, should be cleared.

25.3.4.23 Transmit Buffer Descriptor Ring Start Register (ETDSR)

The ETDSR is written by the user. It provides a pointer to the start of the circular transmit buffer descriptor queue in external memory. This pointer must be 32-bit aligned; however, it is recommended it be made 128-bit aligned (evenly divisible by 16). Bits 30 and 31 should be written to 0 by the user. Non-zero values in these two bit positions are ignored by the hardware.

This register is not reset and must be initialized by the user prior to operation.

Offset: FEC_BASE + 0x0184 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	X_DES_START															
Reset	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	X_DES_START														0	0
Reset	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U

Figure 25-24. Transmit Buffer Descriptor Ring Start Register (ETDSR)

Table 25-26. ETDSR Field Descriptions

Field	Descriptions
X_DES_START	Pointer to start of transmit buffer descriptor queue.
30–31	Reserved, should be cleared.

25.3.4.24 Receive Buffer Size Register (EMRBR)

The EMRBR is a 32-bit register with one 7-bit field programmed by the user. The EMRBR register dictates the maximum size of all receive buffers. Note that because receive frames are truncated at 2 KB – 1 bytes, only bits 21–27 are used. This value should take into consideration that the receive CRC is always written into the last receive buffer. To allow one maximum size frame per buffer, EMRBR must be set to RCR[MAX_FL] or larger. The EMRBR must be evenly divisible by 16. To ensure this, bits 28-31 are forced low. To minimize bus utilization (descriptor fetches) it is recommended that EMRBR be greater than or equal to 256 bytes.

The EMRBR register does not reset, and must be initialized by the user.

Offset: FEC_BASE + 0x0188 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	R_BUF_SIZE							0	0	0	0
W																
Reset	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U

Figure 25-25. Receive Buffer Size Register (EMRBR)

Table 25-27. EMRBR Field Descriptions

Field	Descriptions
0–20	Reserved, should be written to 0 by the host processor.
R_BUF_SIZE	Receive buffer size.
28–31	Reserved, should be written to 0 by the host processor.

25.4 Functional Description

This section describes the operation of the FEC, beginning with the hardware and software initialization sequence, then the software (Ethernet driver) interface for transmitting and receiving frames.

Following the software initialization and operation sections are sections providing a detailed description of the functions of the FEC.

25.4.1 Initialization Sequence

This section describes which registers are reset due to hardware reset, which are reset by the FEC RISC, and what locations the user must initialize prior to enabling the FEC.

25.4.1.1 Hardware Controlled Initialization

In the FEC, registers and control logic that generate interrupts are reset by hardware. A hardware reset deasserts output signals and resets general configuration bits.

Other registers reset when the ECR[ETHER_EN] bit is cleared. ECR[ETHER_EN] is deasserted by a hard reset or may be deasserted by software to halt operation. By deasserting ECR[ETHER_EN], the configuration control registers such as the TCR and RCR are not reset, but the entire data path is reset.

Table 25-28. ECR[ETHER_EN] De-Assertion Effect on FEC

Register/Machine	Reset Value
XMIT block	Transmission is aborted (bad CRC appended)
RECV block	Receive activity is aborted
DMA block	All DMA activity is terminated
RDAR	Cleared
TDAR	Cleared
Descriptor Controller block	Halt operation

25.4.2 User Initialization (Prior to Asserting ECR[ETHER_EN])

The user needs to initialize portions of the FEC prior to setting the ECR[ETHER_EN] bit. The exact values depend on the particular application. The sequence is not important.

Ethernet MAC registers requiring initialization are defined in Table 25-29.

Table 25-29. User Initialization (Before ECR[ETHER_EN])

Description
Initialize EIMR
Clear EIR (write 0xFFFF_FFFF)
TFWR (optional)
IALR / IAUR
GAUR / GALR
PALR / PAUR (only needed for full duplex flow control)
OPD (only needed for full duplex flow control)
RCR
TCR
MSCR (optional)
Clear MIB_RAM (locations Base + 0x0200 – 0x02FC)

FEC FIFO/DMA registers that require initialization are defined in Table 25-30.

Table 25-30. FEC User Initialization (Before ECR[ETHER_EN])

Description
Initialize FRSR (optional)
Initialize EMRBR
Initialize ERDSR
Initialize ETDSR
Initialize (Empty) Transmit Descriptor ring
Initialize (Empty) Receive Descriptor ring

25.4.3 Microcontroller Initialization

In the FEC, the descriptor control RISC initializes some registers after ECR[ETHER_EN] is asserted. After the microcontroller initialization sequence is complete, the hardware is ready for operation.

Table 25-31 shows microcontroller initialization operations.

Table 25-31. Microcontroller Initialization

Description
Initialize BackOff Random Number Seed
Activate Receiver
Activate Transmitter
Clear Transmit FIFO
Clear Receive FIFO
Initialize Transmit Ring Pointer
Initialize Receive Ring Pointer
Initialize FIFO Count Registers

25.4.4 User Initialization (After Asserting ECR[ETHER_EN])

After asserting ECR[ETHER_EN], the user can set up the buffer/frame descriptors and write to the TDAR and RDAR. Refer to Section 25.5, Buffer Descriptors, for more details.

25.4.5 Network Interface Options

The FEC supports both an MII interface for 10/100 Mbps Ethernet and a 7-wire serial interface for 10 Mbps Ethernet. The interface mode is selected by the RCR[MII_MODE] bit. In MII mode (RCR[MII_MODE] = 1), there are 18 signals defined by the IEEE 802.3 standard and supported by the EMAC. These signals are shown in Table 25-32 below.

Table 25-32. MII Mode

Signal Description	EMAC Signal
Transmit Clock	FEC_TX_CLK
Transmit Enable	FEC_TX_EN
Transmit Data	FEC_TXD[3:0]
Transmit Error	FEC_TX_ER
Collision	FEC_COL
Carrier Sense	FEC_CRS
Receive Clock	FEC_RX_CLK
Receive Data Valid	FEC_RX_DV
Receive Data	FEC_RXD[3:0]
Receive Error	FEC_RX_ER
Management Data Clock	FEC_MDC
Management Data Input/Output	FEC_MDIO

The 7-wire serial mode interface (RCR[MII_MODE] = 0) operates in what is generally referred to as the “AMD” mode. 7-wire mode connections to the external transceiver are shown in Table 25-33.

Table 25-33. 7-Wire Mode Configuration

Signal Description	FEC Signal
Transmit Clock	FEC_TX_CLK
Transmit Enable	FEC_TX_EN
Transmit Data	FEC_TXD0
Collision	FEC_COL
Receive Clock	FEC_RX_CLK
Receive Data Valid	FEC_RX_DV
Receive Data	FEC_RXD0

25.4.6 FEC Frame Transmission

The Ethernet transmitter is designed to work with almost no intervention from software. Once ECR[ETHER_EN] is asserted and data appears in the transmit FIFO, the Ethernet MAC is able to transmit onto the network.

When the transmit FIFO fills to the watermark (defined by the TFWR), the MAC transmit logic asserts FEC_TX_EN and starts transmitting the preamble (PA) sequence, the start frame delimiter (SFD), and then the frame information from the FIFO. However, the controller defers the transmission if the network

is busy (FEC_CRS asserts). Before transmitting, the controller waits for carrier sense to become inactive, then determines if carrier sense stays inactive for 60 bit times. If so, the transmission begins after waiting an additional 36 bit times (96 bit times after carrier sense originally became inactive). See Section 25.4.14.1, Transmission Errors, for more details.

If a collision occurs during transmission of the frame (half duplex mode), the Ethernet controller follows the specified backoff procedures and attempts to retransmit the frame until the retry limit is reached. The transmit FIFO stores at least the first 64 bytes of the transmit frame, so that they do not have to be retrieved from system memory in case of a collision. This improves bus utilization and latency in case immediate retransmission is necessary.

When all the frame data has been transmitted, the FCS (frame check sequence or 32-bit cyclic redundancy check, CRC) bytes are appended if the TC bit is set in the transmit frame control word. If the ABC bit is set in the transmit frame control word, a bad CRC is appended to the frame data regardless of the TC bit value. Following the transmission of the CRC, the Ethernet controller writes the frame status information to the MIB block. Short frames are automatically padded by the transmit logic (if the TC bit in the transmit buffer descriptor for the end of frame buffer = 1).

Both buffer (TXB) and frame (TFINT) interrupts may be generated as determined by the settings in the EIMR.

The transmit error interrupts are HBERR, BABT, LATE_COL, COL_RETRY_LIM, and XFIFO_UN. If the transmit frame length exceeds MAX_FL bytes, the BABT interrupt is asserted but the entire frame is transmitted (no truncation).

To pause transmission, set the GTS (graceful transmit stop) bit in the TCR register. When the TCR[GTS] is set, the FEC transmitter stops immediately if transmission is not in progress; otherwise, it continues transmission until the current frame either finishes or terminates with a collision. After the transmitter has stopped, the GRA (graceful stop complete) interrupt is asserted. If TCR[GTS] is cleared, the FEC resumes transmission with the next frame.

The Ethernet controller transmits bytes least significant bit first.

NOTE

At certain cases, the Fast Ethernet Controller (FEC) will transmit single frames more than once. The FEC fetches the transmit buffer descriptors (TxBDs) and the corresponding Tx data continuously until the Tx FIFO is full. It does not determine whether the TxBD to be fetched is already being processed internally (as a result of a wrap). As the FEC nears the end of the transmission of one frame, it begins to DMA the data for the next frame. To remain one BD ahead of the DMA, it also fetches the TxBD for the next frame. The FEC may fetch a BD from memory that has already been processed but not yet written back (it is read a second time with the R bit still set). In this case, the data is fetched and transmitted again. Using at least three TxBDs fixes this problem for large frames, but not for small frames. To ensure correct operation for large or small frames, one of the following must be true:

- The FEC software driver ensures that the Ready bit cleared in at least one TxBD.
- Every frame uses more than one TxBD and every TxBD but the last is written back immediately after the data is fetched.
- The FEC software driver ensures a minimum frame size, n. The minimum number of TxBDs is then rounded up to the nearest integer (although the result cannot be less than 3). The default Tx FIFO size is 192 Bytes; this size is programmable.

25.4.7 FEC Frame Reception

The FEC receiver is designed to work with almost no intervention from the host and can perform address recognition, CRC checking, short frame checking, and maximum frame length checking.

When the driver enables the FEC receiver by asserting ECR[ETHER_EN], it starts processing receive frames immediately. When FEC_RX_DV asserts, the receiver first checks for a valid PA/SFD header. If the PA/SFD is valid, it is stripped and the frame processed by the receiver. If a valid PA/SFD is not found, the frame is ignored.

In serial mode, the first 16 bit times of FEC_RXD0 following assertion of FEC_RX_DV are ignored. Following the first 16 bit times the data sequence is checked for alternating 1/0s. If a 11 or 00 data sequence is detected during bit times 17 to 21, the remainder of the frame is ignored. After bit time 21, the data sequence is monitored for a valid SFD (11). If a 00 is detected, the frame is rejected. When a 11 is detected, the PA/SFD sequence is complete.

In MII mode, the receiver checks for at least one byte matching the SFD. Zero or more PA bytes may occur, but if a 00 bit sequence is detected prior to the SFD byte, the frame is ignored.

After the first 6 bytes of the frame have been received, the FEC performs address recognition on the frame.

Once a collision window (64 bytes) of data has been received and if address recognition has not rejected the frame, the receive FIFO is signalled that the frame is “accepted” and may be passed on to the DMA. If the frame is a runt (due to collision) or is rejected by address recognition, the receive FIFO is notified to “reject” the frame. Thus, no collision fragments are presented to the user except late collisions, which indicate serious LAN problems.

During reception, the Ethernet controller checks for various error conditions and once the entire frame is written into the FIFO, a 32-bit frame status word is written into the FIFO. This status word contains the M, BC, MC, LG, NO, CR, OV and TR status bits, and the frame length. See Section 25.4.14.2, Reception Errors, for more details.

Receive buffer (RXB) and frame interrupts (RFINT) may be generated if enabled by the EIMR register. A receive error interrupt is babbling receiver error (BABR). Receive frames are not truncated if they exceed the max frame length (MAX_FL); however, the BABR interrupt occurs and the LG bit in the receive buffer descriptor (RxBD) is set. See Section 25.5.2, Ethernet Receive Buffer Descriptor (RxBD), for more details.

When the receive frame is complete, the FEC sets the L-bit in the RxBD, writes the other frame status bits into the RxBD, and clears the E-bit. The Ethernet controller next generates a maskable interrupt (RFINT

bit in EIR, maskable by RFIEN bit in EIMR), indicating that a frame has been received and is in memory. The Ethernet controller then waits for a new frame.

The Ethernet controller receives serial data LSB first.

25.4.8 Ethernet Address Recognition

The FEC filters the received frames based on the type of destination address (DA)—individual (unicast), group (multicast), or broadcast (all-ones group address). The difference between an individual address and a group address is determined by the I/G bit (bit 40) in the destination address field. A flowchart for address recognition on received frames is illustrated in the figures below.

Address recognition is accomplished through the use of the receive block and microcode running on the microcontroller. The flowchart shown in Figure 25-26 illustrates the address recognition decisions made by the receive block, while Figure 25-27 illustrates the decisions made by the microcontroller.

If the DA is a broadcast address and broadcast reject (RCR[BC_REJ]) is deasserted, then the frame is accepted unconditionally, as shown in Figure 25-26. Otherwise, if the DA is not a broadcast address, then the microcontroller runs the address recognition subroutine, as shown in Figure 25-27.

If the DA is a group (multicast) address and flow control is disabled, then the microcontroller performs a group hash table lookup using the 64-entry hash table programmed in GAUR and GALR. If a hash match occurs, the receiver accepts the frame.

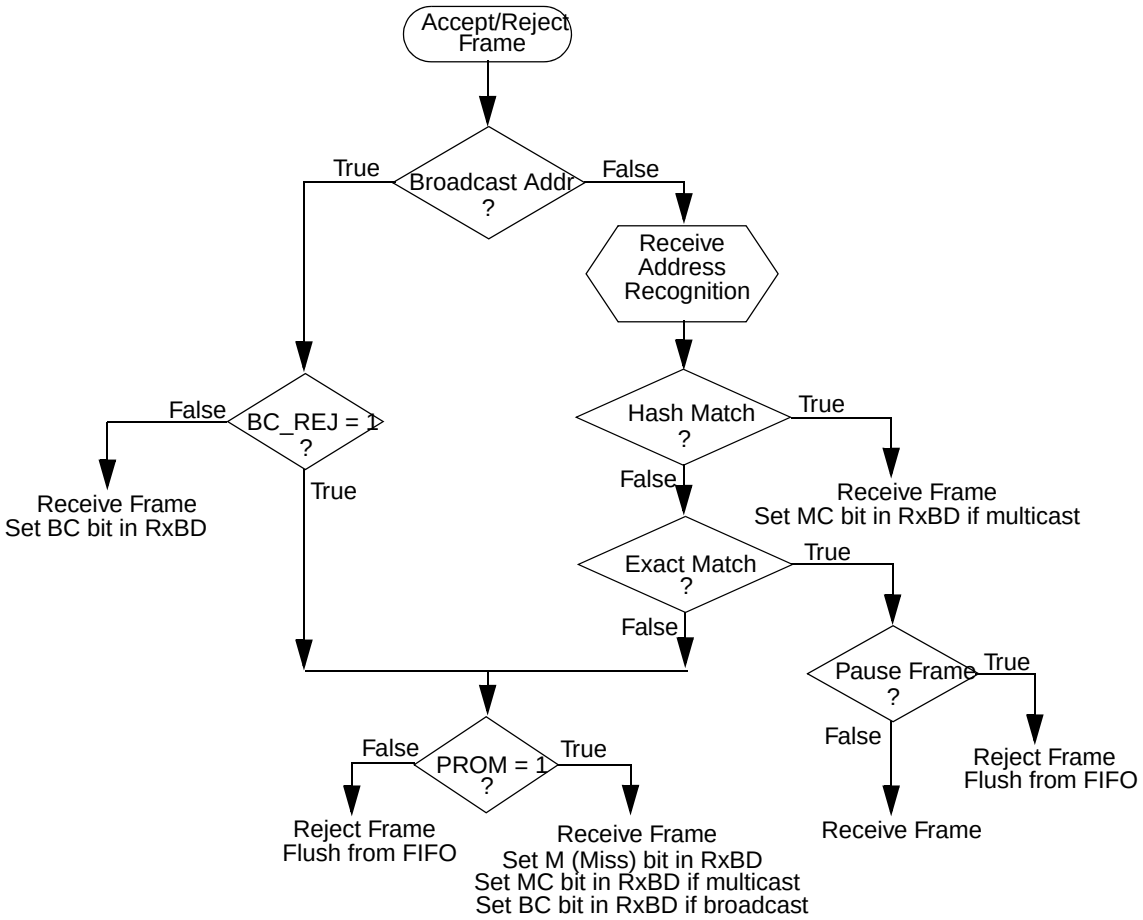
If flow control is enabled, the microcontroller does an exact address match check between the DA and the designated PAUSE DA (01:80:C2:00:00:01). If the receive block determines that the received frame is a valid PAUSE frame, then the frame is rejected. Note the receiver detects a PAUSE frame with the DA field set to either the designated PAUSE DA or to the unicast physical address.

If the DA is the individual (unicast) address, the microcontroller performs an individual exact match comparison between the DA and 48-bit physical address that the user programs in the PALR and PAUR registers. If an exact match occurs, the frame is accepted; otherwise, the microcontroller does an individual hash table lookup using the 64-entry hash table programmed in registers, IAUR and IALR. In the case of an individual hash match, the frame is accepted. Again, the receiver accepts or rejects the frame based on PAUSE frame detection, shown in Figure 25-26.

If neither a hash match (group or individual), nor an exact match (group or individual) occur, then if promiscuous mode is enabled (RCR[PROM] = 1), then the frame is accepted and the MISS bit in the receive buffer descriptor is set; otherwise, the frame is rejected.

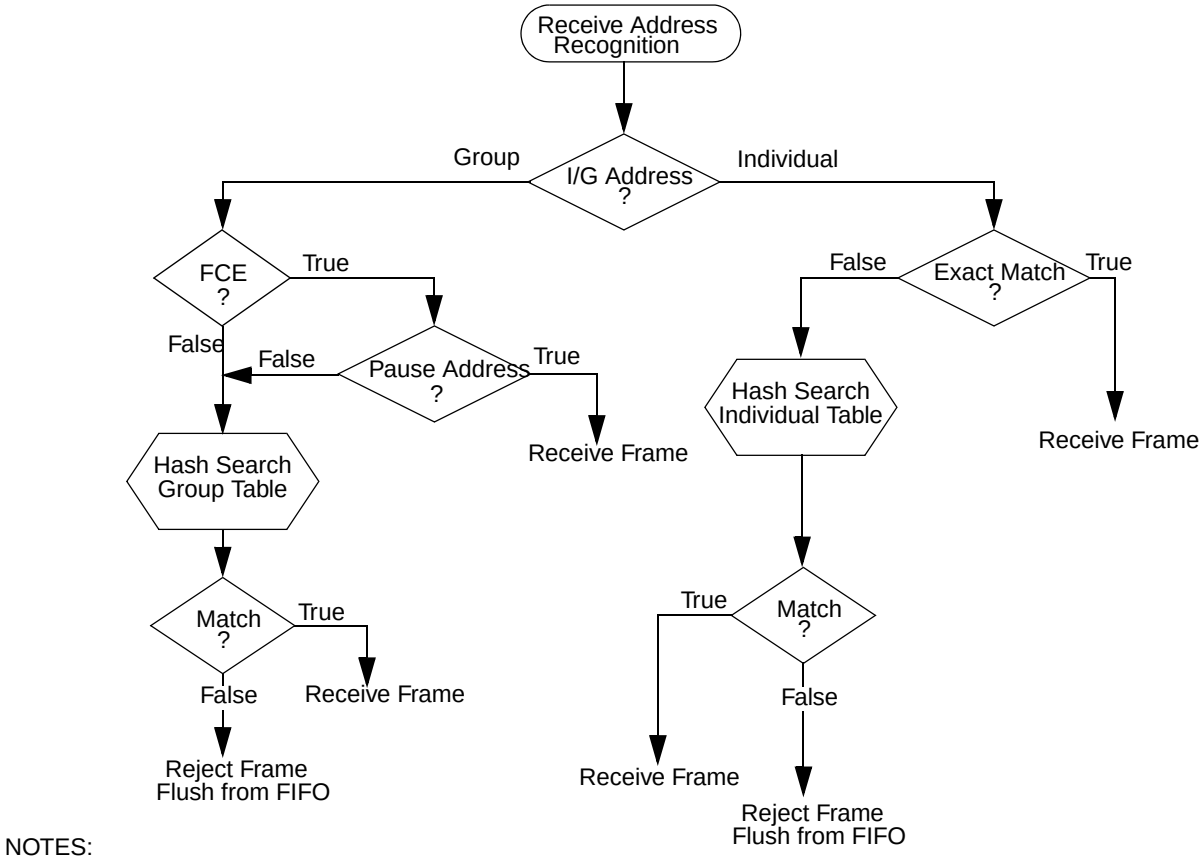
Similarly, if the DA is a broadcast address, broadcast reject (RCR[BC_REJ]) is asserted, and promiscuous mode is enabled, then the frame is accepted and the MISS bit in the receive buffer descriptor is set; otherwise, the frame is rejected.

In general, when a frame is rejected, it is flushed from the FIFO.



NOTES:
BC_REJ - field in RCR register (BroadCast REJect)
PROM - field in RCR register (PROMiscuous mode)
Pause Frame - valid Pause frame received

Figure 25-26. Ethernet Address Recognition—Receive Block Decisions



NOTES:
FCE - field in RCR register (Flow Control Enable)
I/G - Individual/Group bit in Destination Address (least significant bit in first byte received in MAC frame)

Figure 25-27. Ethernet Address Recognition—Microcode Decisions

25.4.9 Hash Algorithm

The hash table algorithm used in the group and individual hash filtering operates as follows. The 48-bit destination address is mapped into one of 64 bits, which are represented by 64 bits stored in GAUR, GALR (group address hash match) or IAUR, IALR (individual address hash match). This mapping is performed by passing the 48-bit address through the on-chip 32-bit CRC generator and selecting the 6 most significant bits of the CRC-encoded result to generate a number between 0 and 63. The MSB of the CRC result selects GAUR (MSB = 1) or GALR (MSB = 0). The least significant 5 bits of the hash result select the bit within the selected register. If the CRC generator selects a bit that is set in the hash table, the frame is accepted; otherwise, it is rejected.

For example, if eight group addresses are stored in the hash table and random group addresses are received, the hash table prevents roughly 56/64 (or 87.5%) of the group address frames from reaching memory. Those that do reach memory must be further filtered by the processor to determine if they truly contain one of the eight desired addresses.

The effectiveness of the hash table declines as the number of addresses increases.

The hash table registers must be initialized by the user. The CRC32 polynomial to use in computing the hash is:

$$X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^{8} + X^{7} + X^{5} + X^{4} + X^{2} + X + 1$$

A table of example Destination Addresses and corresponding hash values is included below for reference.

Table 25-34. Destination Address to 6-Bit Hash

48-bit DA	6-bit Hash (in hex)	Hash Decimal Value
65:ff:ff:ff:ff:ff	0x0	0
55:ff:ff:ff:ff:ff	0x1	1
15:ff:ff:ff:ff:ff	0x2	2
35:ff:ff:ff:ff:ff	0x3	3
B5:ff:ff:ff:ff:ff	0x4	4
95:ff:ff:ff:ff:ff	0x5	5
D5:ff:ff:ff:ff:ff	0x6	6
F5:ff:ff:ff:ff:ff	0x7	7
DB:ff:ff:ff:ff:ff	0x8	8
FB:ff:ff:ff:ff:ff	0x9	9
BB:ff:ff:ff:ff:ff	0xA	10
8B:ff:ff:ff:ff:ff	0xB	11
0B:ff:ff:ff:ff:ff	0xC	12
3B:ff:ff:ff:ff:ff	0xD	13
7B:ff:ff:ff:ff:ff	0xE	14
5B:ff:ff:ff:ff:ff	0xF	15
27:ff:ff:ff:ff:ff	0x10	16
07:ff:ff:ff:ff:ff	0x11	17
57:ff:ff:ff:ff:ff	0x12	18
77:ff:ff:ff:ff:ff	0x13	19
F7:ff:ff:ff:ff:ff	0x14	20
C7:ff:ff:ff:ff:ff	0x15	21
97:ff:ff:ff:ff:ff	0x16	22
A7:ff:ff:ff:ff:ff	0x17	23
99:ff:ff:ff:ff:ff	0x18	24
B9:ff:ff:ff:ff:ff	0x19	25
F9:ff:ff:ff:ff:ff	0x1A	26

Table 25-34. Destination Address to 6-Bit Hash (continued)

48-bit DA	6-bit Hash (in hex)	Hash Decimal Value
C9:ff:ff:ff:ff:ff	0x1B	27
59:ff:ff:ff:ff:ff	0x1C	28
79:ff:ff:ff:ff:ff	0x1D	29
29:ff:ff:ff:ff:ff	0x1E	30
19:ff:ff:ff:ff:ff	0x1F	31
D1:ff:ff:ff:ff:ff	0x20	32
F1:ff:ff:ff:ff:ff	0x21	33
B1:ff:ff:ff:ff:ff	0x22	34
91:ff:ff:ff:ff:ff	0x23	35
11:ff:ff:ff:ff:ff	0x24	36
31:ff:ff:ff:ff:ff	0x25	37
71:ff:ff:ff:ff:ff	0x26	38
51:ff:ff:ff:ff:ff	0x27	39
7F:ff:ff:ff:ff:ff	0x28	40
4F:ff:ff:ff:ff:ff	0x29	41
1F:ff:ff:ff:ff:ff	0x2A	42
3F:ff:ff:ff:ff:ff	0x2B	43
BF:ff:ff:ff:ff:ff	0x2C	44
9F:ff:ff:ff:ff:ff	0x2D	45
DF:ff:ff:ff:ff:ff	0x2E	46
EF:ff:ff:ff:ff:ff	0x2F	47
93:ff:ff:ff:ff:ff	0x30	48
B3:ff:ff:ff:ff:ff	0x31	49
F3:ff:ff:ff:ff:ff	0x32	50
D3:ff:ff:ff:ff:ff	0x33	51
53:ff:ff:ff:ff:ff	0x34	52
73:ff:ff:ff:ff:ff	0x35	53
23:ff:ff:ff:ff:ff	0x36	54
13:ff:ff:ff:ff:ff	0x37	55
3D:ff:ff:ff:ff:ff	0x38	56
0D:ff:ff:ff:ff:ff	0x39	57
5D:ff:ff:ff:ff:ff	0x3A	58

Table 25-34. Destination Address to 6-Bit Hash (continued)

48-bit DA	6-bit Hash (in hex)	Hash Decimal Value
7D:ff:ff:ff:ff:ff	0x3B	59
FD:ff:ff:ff:ff:ff	0x3C	60
DD:ff:ff:ff:ff:ff	0x3D	61
9D:ff:ff:ff:ff:ff	0x3E	62
BD:ff:ff:ff:ff:ff	0x3F	63

25.4.10 Full Duplex Flow Control

Full-duplex flow control allows the user to transmit pause frames and to detect received pause frames. Upon detection of a pause frame, MAC data frame transmission stops for a given pause duration.

To enable pause frame detection, the FEC must operate in full-duplex mode (TCR[FDEN] asserted) and flow control enable (RCR[FCE]) must be asserted. The FEC detects a pause frame when the fields of the incoming frame match the pause frame specifications, as shown in the table below. In addition, the receive status associated with the frame should indicate that the frame is valid.

Table 25-35. PAUSE Frame Field Specification

48-bit Destination Address	0x0180_C200_0001 or Physical Address
48-bit Source Address	Any
16-bit TYPE	0x8808
16-bit OPCODE	0x0001
16-bit PAUSE_DUR	0x0000 to 0xFFFF

Pause frame detection is performed by the receiver and microcontroller modules. The microcontroller runs an address recognition subroutine to detect the specified pause frame destination address, while the receiver detects the TYPE and OPCODE pause frame fields. On detection of a pause frame, TCR[GTS] is asserted by the FEC internally. When transmission has paused, the EIR[GRA] interrupt is asserted and the pause timer begins to increment. Note that the pause timer makes use of the transmit backoff timer hardware, which is used for tracking the appropriate collision backoff time in half-duplex mode. The pause timer increments once every slot time, until OPD[PAUSE_DUR] slot times have expired. On OPD[PAUSE_DUR] expiration, TCR[GTS] is deasserted allowing MAC data frame transmission to resume. Note that the receive flow control pause (TCR[RFC_PAUSE]) status bit is asserted while the transmitter is paused due to reception of a pause frame.

To transmit a pause frame, the FEC must operate in full-duplex mode and the user must assert flow control pause (TCR[TFC_PAUSE]). On assertion of transmit flow control pause (TCR[TFC_PAUSE]), the transmitter asserts TCR[GTS] internally. When the transmission of data frames stops, the EIR[GRA] (graceful stop complete) interrupt asserts. Following EIR[GRA] assertion, the pause frame is transmitted. On completion of pause frame transmission, flow control pause (TCR[TFC_PAUSE]) and TCR[GTS] are deasserted internally.

The user must specify the desired pause duration in the OPD register.

Note that when the transmitter is paused due to receiver/microcontroller pause frame detection, transmit flow control pause (TCR[TFC_PAUSE]) may still be asserted and causes the transmission of a single pause frame. In this case, the EIR[GRA] interrupt is not asserted.

25.4.11 Inter-Packet Gap (IPG) Time

The minimum inter-packet gap time for back-to-back transmission is 96 bit times. After completing a transmission or after the backoff algorithm completes, the transmitter waits for carrier sense to be negated before starting its 96 bit time IPG counter. Frame transmission may begin 96 bit times after carrier sense is negated if it stays negated for at least 60 bit times. If carrier sense asserts during the last 36 bit times, it is ignored and a collision occurs.

The receiver receives back-to-back frames with a minimum spacing of at least 28 bit times. If an inter-packet gap between receive frames is less than 28 bit times, the following frame may be discarded by the receiver.

25.4.12 Collision Handling

If a collision occurs during frame transmission, the Ethernet controller continues the transmission for at least 32 bit times, transmitting a JAM pattern consisting of 32 ones. If the collision occurs during the preamble sequence, the JAM pattern is sent after the end of the preamble sequence.

If a collision occurs within 512 bit times, the retry process is initiated. The transmitter waits a random number of slot times. A slot time is 512 bit times. If a collision occurs after 512 bit times, then no retransmission is performed and the end of frame buffer is closed with a late collision (LC) error indication.

25.4.13 Internal and External Loopback

Both internal and external loopback are supported by the Ethernet controller. In loopback mode, both of the FIFOs are used and the FEC actually operates in a full-duplex fashion. Both internal and external loopback are configured using combinations of the LOOP and DRT bits in the RCR register and the FDEN bit in the TCR register.

For both internal and external loopback set FDEN = 1.

For internal loopback, set RCR[LOOP] = 1 and RCR[DRT] = 0. FEC_TX_EN and FEC_TX_ER do not assert during internal loopback. During internal loopback, the transmit/receive data rate is higher than in normal operation because the internal system clock is used by the transmit and receive blocks instead of the clocks from the external transceiver. This causes an increase in the required system bus bandwidth for transmit and receive data being DMA’d to/from external memory. It may be necessary to pace the frames on the transmit side and/or limit the size of the frames to prevent transmit FIFO underrun and receive FIFO overflow.

For external loopback set RCR[LOOP] = 0, RCR[DRT] = 0 and configure the external transceiver for loopback.

25.4.14 Ethernet Error-Handling Procedure

The Ethernet controller reports frame reception and transmission error conditions using the FEC RxBDs, the EIR register, and the MIB block counters.

25.4.14.1 Transmission Errors

25.4.14.1.1 Transmitter Underrun

If this error occurs, the FEC sends 32 bits that ensure a CRC error and stops transmitting. All remaining buffers for that frame are then flushed and closed. The UN bit is set in the EIR. The FEC then continues to the next transmit buffer descriptor and begins transmitting the next frame.

The “UN” interrupt is asserted if enabled in the EIMR register.

25.4.14.1.2 Retransmission Attempts Limit Expired

When this error occurs, the FEC terminates transmission. All remaining buffers for that frame are flushed and closed, and the RL bit is set in the EIR. The FEC then continues to the next transmit buffer descriptor and begins transmitting the next frame.

The “RL” interrupt is asserted if enabled in the EIMR register.

25.4.14.1.3 Late Collision

When a collision occurs after the slot time (512 bits starting at the preamble), the FEC terminates transmission. All remaining buffers for that frame are flushed and closed, and the LC bit is set in the EIR register. The FEC then continues to the next transmit buffer descriptor and begins transmitting the next frame.

The “LC” interrupt is asserted if enabled in the EIMR register.

25.4.14.1.4 Heartbeat

Some transceivers have a self-test feature called ‘heartbeat’ or ‘signal quality error.’ To signify a good self-test, the transceiver indicates a collision to the FEC within 4 microseconds after completion of a frame transmitted by the Ethernet controller. This indication of a collision does not imply a real collision error on the network, but is rather an indication that the transceiver still seems to be functioning properly. This is called the heartbeat condition.

If the HBC bit is set in the TCR register and the heartbeat condition is not detected by the FEC after a frame transmission, then a heartbeat error occurs. When this error occurs, the FEC closes the buffer, sets the HB bit in the EIR register, and generates the HBERR interrupt if it is enabled.

25.4.14.2 Reception Errors

25.4.14.2.1 Overrun Error

If the receive block has data to put into the receive FIFO and the receive FIFO is full, the FEC sets the OV bit in the RxBD. All subsequent data in the frame is discarded. Subsequent frames may also be discarded until the receive FIFO is serviced by the DMA and space is made available. At this point the receive frame/status word is written into the FIFO with the OV bit set. This frame must be discarded by the driver.

25.4.14.2.2 Non-Octet Error (Dribbling Bits)

The Ethernet controller handles as many as 7 dribbling bits when the receive frame terminates past an non-octet aligned boundary. Dribbling bits are not used in the CRC calculation. If there is a CRC error, then the frame non-octet aligned (NO) error is reported in the RxBD. If there is no CRC error, then no error is reported.

25.4.14.2.3 CRC Error

When a CRC error occurs with no dribble bits, the FEC closes the buffer and sets the CR bit in the RxBD. CRC checking cannot be disabled, but the CRC error can be ignored if checking is not required.

25.4.14.2.4 Frame Length Violation

When the receive frame length exceeds MAX_FL bytes, the BABR interrupt is generated, and the LG bit in the end of frame RxBD is set. The frame is not truncated unless the frame length exceeds 2047 bytes).

25.4.14.2.5 Truncation

When the receive frame length exceeds 2047 bytes the frame is truncated, and the TR bit is set in the RxBD.

25.5 Buffer Descriptors

This section provides a description of the operation of the driver/DMA via the buffer descriptors. It is followed by a detailed description of the receive and transmit descriptor fields.

25.5.1 Driver/DMA Operation with Buffer Descriptors

The data for the FEC frames must reside in memory external to the FEC. The data for a frame is placed in one or more buffers. Associated with each buffer is a buffer descriptor (BD) which contains a starting address (pointer), data length, and status/control information (which contains the current state for the buffer). To permit maximum user flexibility, the BDs are also located in external memory and are read in by the FEC DMA engine.

Software “produces” buffers by allocating/initializing memory and initializing buffer descriptors. Setting the RxBD[E] or TxBD[R] bit “produces” the buffer. Software writing to either the TDAR or RDAR tells the FEC that a buffer has been placed in external memory for the transmit or receive data traffic, respectively. The hardware reads the BDs and “consumes” the buffers after they have been produced. After

the data DMA is complete and the buffer descriptor status bits have been written by the DMA engine, the RxBD[E] or TxBD[R] bit is cleared by hardware to signal that the buffer has been “consumed.” Software may poll the BDs to detect when the buffers have been consumed or may rely on the buffer/frame interrupts. These buffers may then be processed by the driver and returned to the free list.

The ECR[ETHER_EN] signal operates as a reset to the BD/DMA logic. When ECR[ETHER_EN] is deasserted the DMA engine BD pointers are reset to point to the starting transmit and receive BDs. The buffer descriptors are not initialized by hardware during reset. At least one transmit and receive buffer descriptor must be initialized by software before the ECR[ETHER_EN] bit is set.

The buffer descriptors operate as two separate rings. ERDSR defines the starting address for receive BDs and ETDSR defines the starting address for transmit BDs. The last buffer descriptor in each ring is defined by the wrap (W) bit. When set, W indicates that the next descriptor in the ring is at the location pointed to by ERDSR and ETDSR for the receive and transmit rings, respectively. Buffer descriptor rings must start on a 32-bit boundary; however, it is recommended they are made 128-bit aligned.

25.5.1.1 Driver/DMA Operation with Transmit BDs

Typically a transmit frame is divided between multiple buffers. An example is to have an application payload in one buffer, TCP header in a second buffer, IP header in a third buffer, Ethernet/IEEE 802.3 header in a fourth buffer. The Ethernet MAC does not prepend the Ethernet header (destination address, source address, length/type fields), so this must be provided by the driver in one of the transmit buffers. The Ethernet MAC can append the Ethernet CRC to the frame. Whether the CRC is appended by the MAC or by the driver is determined by the TC bit in the transmit BD which must be set by the driver.

The driver (TxBD software producer) should set up Tx BDs in such a way that a complete transmit frame is given to the hardware at once. If a transmit frame consists of three buffers, the BDs should be initialized with pointer, length and control (W, L, TC, ABC) and then the TxBD[R] bits should be set = 1 in reverse order (3rd, 2nd, 1st BD) to ensure that the complete frame is ready in memory before the DMA begins. If the TxBDs are set up in order, the DMA Controller could DMA the first BD before the 2nd was made available, potentially causing a transmit FIFO underrun.

In the FEC, the DMA is notified by the driver that new transmit frames are available by writing to the TDAR register. When this register is written to (data value is not significant) the FEC RISC tells the DMA to read the next transmit BD in the ring. Once started, the RISC + DMA continues to read and interpret transmit BDs in order and DMA the associated buffers, until a transmit BD is encountered with the R bit = 0. At this point, the FEC polls this BD one more time. If the R bit = 0 the second time, then the RISC stops the transmit descriptor read process until software sets up another transmit frame and writes to TDAR.

When the DMA of each transmit buffer is complete, the DMA writes back to the BD to clear the R bit, indicating that the hardware consumer is finished with the buffer.

25.5.1.2 Driver/DMA Operation with Receive BDs

Unlike transmit, the length of the receive frame is unknown by the driver ahead of time. Therefore the driver must set a variable to define the length of all receive buffers. In the FEC, this variable is written to the EMRBR register.

The driver (RxBD software producer) should set up some number of “empty” buffers for the Ethernet by initializing the address field and the E and W bits of the associated receive BDs. The hardware (receive DMA) consumes these buffers by filling them with data as frames are received and clearing the E bit and writing to the L (1 indicates last buffer in frame) bit, the frame status bits (if L = 1), and the length field.

If a receive frame spans multiple receive buffers, the L bit is only set for the last buffer in the frame. For non-last buffers, the length field in the receive BD is written by the DMA (at the same time the E bit is cleared) with the default receive buffer length value. For end of frame buffers, the receive BD is written with L = 1 and information written to the status bits (M, BC, MC, LG, NO, CR, OV, TR). Some of the status bits are error indicators which, if set, indicate the receive frame should be discarded and not given to higher layers. The frame status/length information is written into the receive FIFO following the end of the frame (as a single 32-bit word) by the receive logic. The length field for the end of frame buffer is written with the length of the entire frame, not just the length of the last buffer.

For simplicity the driver may assign the default receive buffer length to be large enough to contain an entire frame, keeping in mind that a malfunction on the network or out of spec implementation could result in giant frames. Frames of 2KB (2048) bytes or larger are truncated by the FEC at 2047 bytes so software is guaranteed never to see a receive frame larger than 2047 bytes.

Similar to transmit, the FEC polls the receive descriptor ring after the driver sets up receive BDs and writes to the RDAR register. As frames are received, the FEC fills receive buffers and updates the associated BDs, then reads the next BD in the receive descriptor ring. If the FEC reads a receive BD and finds the E bit = 0, it polls this BD once more. If the BD = 0 a second time, the FEC stops reading receive BDs until the driver writes to RDAR.

25.5.2 Ethernet Receive Buffer Descriptor (RxBD)

In the RxBD, the user initializes the E and W bits in the first word and the pointer in second word. When the buffer has been DMA’d, the Ethernet controller modifies the E, L, M, BC, MC, LG, NO, CR, OV, and TR bits and writes the length of the used portion of the buffer in the first word. The M, BC, MC, LG, NO, CR, OV and TR bits in the first word of the buffer descriptor are only modified by the Ethernet controller when the L bit is set.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Offset + 0	E	RO1	W	RO2	L	—	—	M	BC	MC	LG	NO	—	CR	OV	TR
Offset + 2	Data Length															
Offset + 4	Tx Data Buffer Pointer - A [0:15]															
Offset + 6	Tx Data Buffer Pointer - A [16:31]															

Figure 25-28. Receive Buffer Descriptor (RxBD)

Table 25-36. Receive Buffer Descriptor Field Definitions

Halfword	Location	Field Name	Description
Offset + 0	Bit 0	E	Empty. Written by the FEC (=0) and user (=1). 0 The data buffer associated with this BD has been filled with received data, or data reception has been aborted due to an error condition. The status and length fields have been updated as required. 1 The data buffer associated with this BD is empty, or reception is currently in progress.
Offset + 0	Bit 1	RO1	Receive software ownership. This field is reserved for use by software. This read/write bit is not modified by hardware, and its value does not affect hardware.
Offset + 0	Bit 2	W	Wrap. Written by user. 0 The next buffer descriptor is found in the consecutive location 1 The next buffer descriptor is found at the location defined in ERDSR.
Offset + 0	Bit 3	RO2	Receive software ownership. This field is reserved for use by software. This read/write bit is not modified by hardware, and its value does not affect hardware.
Offset + 0	Bit 4	L	Last in frame. Written by the FEC. 0 The buffer is not the last in a frame. 1 The buffer is the last in a frame.
Offset + 0	Bits 5-6	—	Reserved.
Offset + 0	Bit 7	M	Miss. Written by the FEC. This bit is set by the FEC for frames that were accepted in promiscuous mode, but were flagged as a “miss” by the internal address recognition. Thus, while in promiscuous mode, the user can use the M-bit to quickly determine whether the frame was destined to this station. This bit is valid only if the L-bit is set and the PROM bit is set. 0 The frame was received because of an address recognition hit. 1 The frame was received because of promiscuous mode.
Offset + 0	Bit 8	BC	Set if the DA is broadcast (FF-FF-FF-FF-FF-FF).
Offset + 0	Bit 9	MC	Set if the DA is multicast and not BC.
Offset + 0	Bit 10	LG	Rx frame length violation. Written by the FEC. A frame length greater than RCR[MAX_FL] was recognized. This bit is valid only if the L-bit is set. The receive data is not altered in any way unless the length exceeds 2047 bytes.
Offset + 0	Bit 11	NO	Receive non-octet aligned frame. Written by the FEC. A frame that contained a number of bits not divisible by 8 was received, and the CRC check that occurred at the preceding byte boundary generated an error. This bit is valid only if the L-bit is set. When this bit is set, the CR bit will not be set.
Offset + 0	Bit 12	—	Reserved.
Offset + 0	Bit 13	CR	Receive CRC error. Written by the FEC. This frame contains a CRC error and is an integral number of octets in length. This bit is valid only if the L-bit is set.
Offset + 0	Bit 14	OV	Overrun. Written by the FEC. A receive FIFO overrun occurred during frame reception. If this bit is set, the other status bits, M, LG, NO, CR, and CL lose their normal meaning and will be zero. This bit is valid only if the L-bit is set.
Offset + 0	Bit 15	TR	Set if the receive frame is truncated (frame length > 2047 bytes). If the TR bit is set the frame should be discarded and the other error bits should be ignored as they may be incorrect.

Table 25-36. Receive Buffer Descriptor Field Definitions (continued)

Halfword	Location	Field Name	Description
Offset + 2	Bits [0:15]	Data Length	Data length. Written by the FEC. Data length is the number of 8-bit data groups (octets) written by the FEC into this BD's data buffer if L = 0 (the value is equal to EMRBR), or the length of the frame including CRC if L = 1. It is written by the FEC once as the BD is closed.
0ffset + 4	Bits [0:15]	A[0:15]]	RX data buffer pointer, bits [0:15][1]
Offset + 6	Bits [0:15]	A[16:31]	RX data buffer pointer, bits [16:31]

1 The receive buffer pointer, which contains the address of the associated data buffer, must always be evenly divisible by 16. The buffer must reside in memory external to the FEC. This value is never modified by the Ethernet controller.

NOTE

Whenever the software driver sets an E bit in one or more receive descriptors, the driver should follow that with a write to RDAR.

25.5.3 Ethernet Transmit Buffer Descriptor (TxBD)

Data is presented to the FEC for transmission by arranging it in buffers referenced by the channel's TxBDs. The Ethernet controller confirms transmission by clearing the ready bit (R bit) when DMA of the buffer is complete. In the TxBD the user initializes the R, W, L, and TC bits and the length (in bytes) in the first word, and the buffer pointer in the second word.

The FEC sets the R bit = 0 in the first word of the BD when the buffer has been DMA'd. Status bits for the buffer/frame are not included in the transmit buffer descriptors. Transmit frame status is indicated via individual interrupt bits (error conditions) and in statistic counters in the MIB block. See Section 25.3.3, MIB Block Counters Memory Map, for more details.

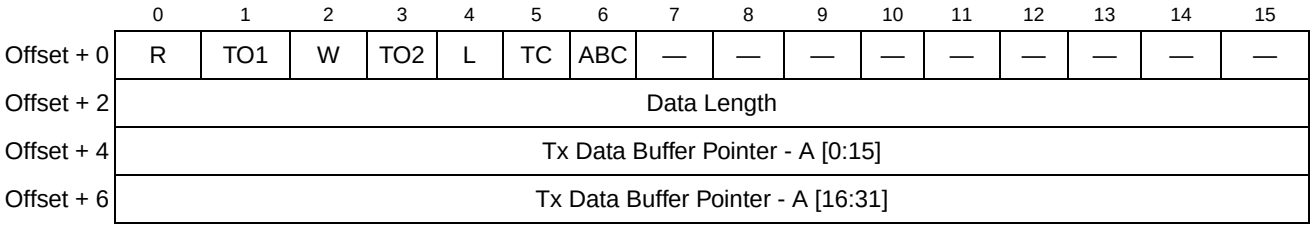


Figure 25-29. Transmit Buffer Descriptor (TxBD)

Table 25-37. Transmit Buffer Descriptor Field Definitions

Halfword	Location	Field Name	Description
Offset + 0	Bit 0	R	Ready. Written by the FEC and the user. 0 The data buffer associated with this BD is not ready for transmission. The user is free to manipulate this BD or its associated data buffer. The FEC clears this bit after the buffer has been transmitted or after an error condition is encountered. 1 The data buffer, which has been prepared for transmission by the user, has not been transmitted or is currently being transmitted. No fields of this BD may be written by the user once this bit is set.
Offset + 0	Bit 1	TO1	Transmit software ownership. This field is reserved for software use. This read/write bit is not modified by hardware, and its value does not affect hardware.
Offset + 0	Bit 2	W	Wrap. Written by user. 0 The next buffer descriptor is found in the consecutive location 1 The next buffer descriptor is found at the location defined in ETDSR.
Offset + 0	Bit 3	TO2	Transmit software ownership. This field is reserved for use by software. This read/write bit is not modified by hardware, and its value does not affect hardware.
Offset + 0	Bit 4	L	Last in frame. Written by user. 0 The buffer is not the last in the transmit frame. 1 The buffer is the last in the transmit frame.
Offset + 0	Bit 5	TC	Tx CRC. Written by user (only valid if L = 1). 0 End transmission immediately after the last data byte. 1 Transmit the CRC sequence after the last data byte.
Offset + 0	Bit 6	ABC	Append bad CRC. Written by user (only valid if L = 1). 0 No effect 1 Transmit the CRC sequence inverted after the last data byte (regardless of TC value).
Offset + 0	Bits [7:15]	—	Reserved.
Offset + 2	Bits [0:15]	Data Length	Data length, written by user. Data length is the number of octets the FEC should transmit from this BD's data buffer. It is never modified by the FEC. Bits [0:10] are used by the DMA engine, bits[11:15] are ignored.
Offset + 4	Bits [0:15]	A[0:15]	Tx data buffer pointer, bits [0:15][1]
Offset + 6	Bits [0:15]	A[16:31]	Tx data buffer pointer, bits [16:31].

[1] The transmit buffer pointer, which contains the address of the associated data buffer, must always be evenly divisible by 4. The buffer must reside in memory external to the FEC. This value is never modified by the Ethernet controller.

NOTE

Once the software driver has set up the buffers for a frame, it should set up the corresponding BDs. The last step in setting up the BDs for a transmit frame should be to set the R bit in the first BD for the frame. The driver should follow that with a write to TDAR, which triggers the FEC to poll the next BD in the ring.

Chapter 26
FlexRay Communication Controller (FlexRAY)

26.1 Introduction

26.1.1 Reference

The following documents are referenced.

- *FlexRay Communications System Protocol Specification, Version 2.1 Rev A*
- *FlexRay Communications System Electrical Physical Layer Specification, Version 2.1 Rev A*

26.1.2 Glossary

This section provides a list of terms used in the description of the controller.

Table 26-1. List of Terms

Term	Definition
BCU	Buffer Control Unit. Handles message buffer access.
BMIF	Bus Master Interface. Provides master access to FlexRay Memory block.
CC	Communication Controller
CDC	Clock Domain Crosser
CHI	Controller Host Interface
Cycle length in μT	The actual length of a cycle in μT for the ideal controller (+/- 0 ppm)
EBI	External Bus Interface
FlexRay Memory	Memory Window to store message buffer payload, header, status, and synchronization frame related tables.
System Memory	Memory that is contains the FlexRay Memory
System Bus	Bus that connects the controller and System Memory
FSS	Frame Start Sequence
HIF	Host Interface. Provides host access to controller.
Host	The FlexRay CC host MCU
LUT	Look Up Table. Stores message buffer header index value.
MB	Message Buffer
MBIDX	Message Buffer Index: the position of a header field entry within the header area. If the header area is accessed as an array, this is the same as the array index of the entry.
MBNum	Message Buffer Number: Position of message buffer configuration registers within the register map. For example, Message Buffer Number 5 corresponds to the MBCCS5 register.
MCU	Microcontroller Unit

Table 26-1. List of Terms (continued)

Term	Definition
μT	Microtick
MT	Macrotick
MTS	Media Access Test Symbol
NIT	Network Idle Time
PE	Protocol Engine
POC	Protocol Operation Control. Each state of the POC is denoted by *POC:state*
Rx	Reception
SEQ	Sequencer Engine
TCU	Time Control Unit
Tx	Transmission
sync frame	null frame or message frame with *Sync Frame Indicator* set to 1
startup frame	null frame or message frame with both *Sync Frame Indicator* and *Startup Frame Indicator* set to 1
normal frame	null frame or message frame with both *Sync Frame Indicator* and *Startup Frame Indicator* set to 0
null frame	frame with *Null Frame Indicator* set to 0
message frame	frame with *Null Frame Indicator* set to 1

26.1.3 Color Coding

Throughout this chapter types of items are highlighted through the use of an italicized color font.

FlexRay protocol parameters, constants and variables are highlighted with *blue italics*. An example is the parameter *gdActionPointOffset*.

FlexRay protocol states are highlighted in *green italics*. An example is the state *POC:normal active*.

26.1.4 Overview

The controller is a FlexRay communication controller that implements the *FlexRay Communications System Protocol Specification, Version 2.1 Rev A.*

The controller has three main components:

- Controller host interface (CHI)
- Protocol engine (PE)
- Clock domain crossing unit (CDC)

A block diagram of the controller with its surrounding modules is given in Figure 26-1.

NOTE

The FlexRay block is not implemented on the PXN21.

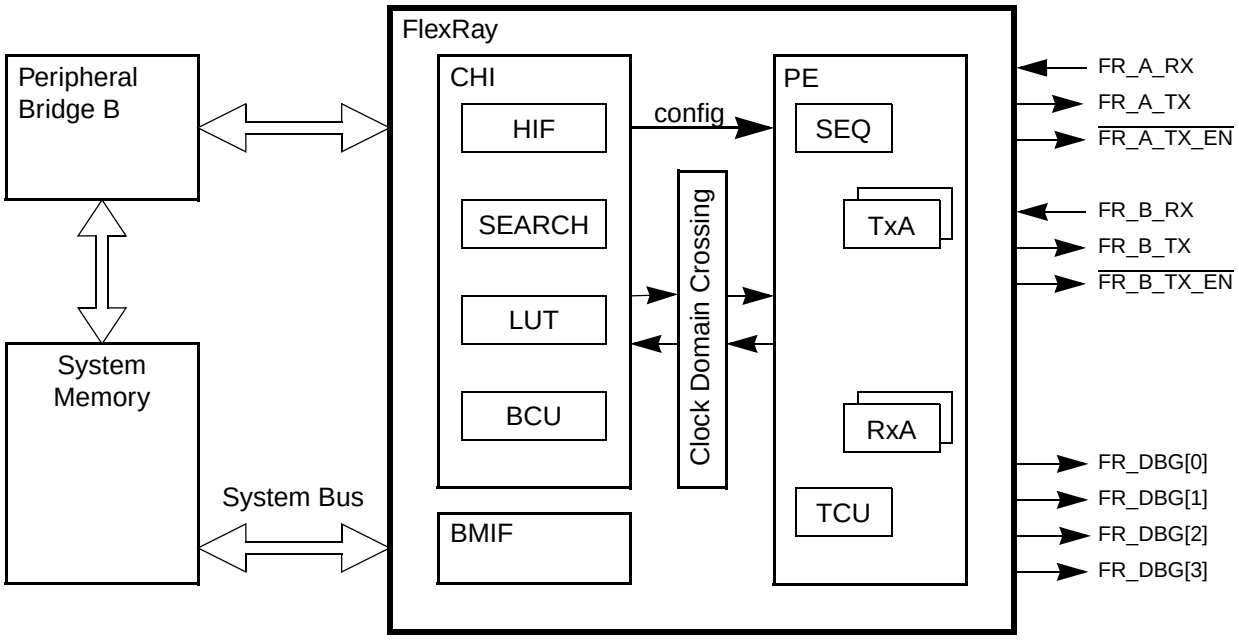


Figure 26-1. FlexRay Block Diagram

The protocol engine has two transmitter units TxA and TxB and two receiver units RxA and RxB for sending and receiving frames through the two FlexRay channels. The time control unit (TCU) is responsible for maintaining global clock synchronization to the FlexRay network. The overall activity of the PE is controlled by the sequencer engine (SEQ).

The controller host interface provides host access to the module's configuration, control, and status registers, as well as to the message buffer configuration, control, and status registers. The message buffers themselves, which contain the frame header and payload data received or to be transmitted, and the slot status information, are stored in the FlexRay memory.

The clock domain crossing unit implements signal crossing from the CHI clock domain to the PE clock domain and vice versa, to allow for asynchronous PE and CHI clock domains.

The controller stores the frame header and payload data of frames received or of frames to be transmitted in the FlexRay memory. The application accesses the FlexRay memory to retrieve and provide the frames to be processed by the controller. In addition to the frame header and payload data, the controller stores the synchronization frame related tables in the FlexRay memory for application processing.

The FlexRay memory is located in the system memory of the MCU. The controller has access to the FlexRay memory via its bus master interface (BMIF). The host provides the start address of the FlexRay memory window within the system memory by programming the System Memory Base Address Register (SYMBADR). All FlexRay memory related offsets are stored in offset registers. The physical address pointer into the FlexRay memory window of the MCU system memory is calculated using the offset values the FlexRay memory base address.

NOTE

The controller does not provide a memory protection scheme for the FlexRay memory.

26.1.5 Features

The controller provides the following features:

- *FlexRay Communications System Protocol Specification, Version 2.1 Rev A* compliant protocol implementation
- *FlexRay Communications System Electrical Physical Layer Specification, Version 2.1 Rev A* compliant bus driver interface
- Single channel support
  - FlexRay Port A can be configured to be connected either to physical FlexRay channel A or physical FlexRay channel B.
- FlexRay bus data rates of 10 Mbit/s, 8 Mbit/s, 5 Mbit/s, and 2.5 Mbit/s supported
- 128 configurable message buffers with
  - Individual frame ID filtering
  - Individual channel ID filtering
  - Individual cycle counter filtering
- Message buffer header, status and payload data stored in dedicated FlexRay memory
  - Allows for flexible and efficient message buffer implementation
  - Consistent data access ensured by means of buffer locking scheme
  - Application can lock multiple buffers at the same time
- Size of message buffer payload data section configurable from 0 to 254 bytes
- Two independent message buffer segments with configurable size of payload data section
  - Each segment can contain message buffers assigned to the static segment and message buffers assigned to the dynamic segment at the same time
- Zero padding for transmit message buffers in static segment
  - Applied when the frame payload length exceeds the size of the message buffer data section
- Transmit message buffers configurable with state/event semantics
- Message buffers can be configured as
  - Receive message buffer
  - Single buffered transmit message buffer
  - Double buffered transmit message buffer (combines two single buffered message buffer)
- Individual message buffer reconfiguration supported
  - Means provided to safely disable individual message buffers
  - Disabled message buffers can be reconfigured
- Two independent receive FIFOs
  - One receive FIFO per channel

  - As many as 255 entries for each FIFO
  - Global frame ID filtering, based on both value/mask filters and range filters
  - Global channel ID filtering
  - Global message ID filtering for the dynamic segment
- Four configurable slot error counters
- Four dedicated slot status indicators
  - Used to observe slots without using receive message buffers
- Measured value indicators for the clock synchronization
  - Internal synchronization frame ID and synchronization frame measurement tables can be copied into the FlexRay memory
- Fractional macroticks are supported for clock correction
- Maskable interrupt sources provided via individual and combined interrupt lines
- One absolute timer
- One timer that can be configured to absolute or relative

26.1.6 Modes of Operation

This section describes the basic operational power modes of the controller.

26.1.6.1 Disabled Mode

The controller enters the Disabled Mode during hard reset. The controller indicates that it is in the Disabled Mode by negating the module enable bit MEN in the Module Configuration Register (MCR).

No communication is performed on the FlexRay bus.

All registers with the write access conditions *Any Time* and *Disabled Mode* can be accessed for writing as stated in Section 26.5.2, Register Descriptions.

The application configures the controller by accessing the configuration bits and fields in the Module Configuration Register (MCR).

26.1.6.1.1 Leave Disabled Mode

The controller leaves the Disabled Mode and enters the Normal Mode, when the application writes 1 to the module enable bit MEN in the Module Configuration Register (MCR).

NOTE

When the controller was enabled, it cannot be disabled later on.

26.1.6.2 Normal Mode

In this mode the controller is fully functional. The controller indicates that it is in Normal Mode by asserting the module enable bit MEN in the Module Configuration Register (MCR).

26.1.6.2.1 Enter Normal Mode

This mode is entered when the application requests the controller to leave the Disabled Mode. If the Normal Mode was entered by leaving the Disabled Mode, the application has to perform the protocol initialization described in Section 26.7.1.2, Protocol Initialization, to achieve full FlexRay functionality.

Depending on the values of the SCM, CHA, and CHB bits in the Module Configuration Register (MCR), the corresponding FlexRay bus driver ports are enabled and driven.

26.2 External Signal Description

This section lists and describes the controller signals, connected to external pins. These signals are summarized in Table 26-2 and described in detail in Section 26.2.1, Detailed Signal Descriptions.

NOTE

The off chip signals FR_A_RX, FR_A_TX, and $\overline{\text{FR\_A\_TX\_EN}}$ are available on each package option. The availability of the other off chip signals depends on the package option.

Table 26-2. External Signal Properties

Name	Direction	Active	Reset	Function
FR_A_RX	Input	—	—	Receive Data Channel A
FR_A_TX	Output	—	1	Transmit Data Channel A
$\overline{\text{FR\_A\_TX\_EN}}$	Output	Low	1	Transmit Enable Channel A
FR_B_RX	Input	—	—	Receive Data Channel B
FR_B_TX	Output	—	1	Transmit Data Channel B
$\overline{\text{FR\_B\_TX\_EN}}$	Output	Low	1	Transmit Enable Channel B
FR_DBG[0]	Output	—	0	Debug Strobe Signal 0
FR_DBG[1]	Output	—	0	Debug Strobe Signal 1
FR_DBG[2]	Output	—	0	Debug Strobe Signal 2
FR_DBG[3]	Output	—	0	Debug Strobe Signal 3

26.2.1 Detailed Signal Descriptions

This section provides a detailed description of the controller signals, connected to external pins.

26.2.1.1 FR_A_RX — Receive Data Channel A

The FR_A_RX signal carries the receive data for channel A from the corresponding FlexRay bus driver.

26.2.1.2 FR_A_TX — Transmit Data Channel A

The FR_A_TX signal carries the transmit data for channel A to the corresponding FlexRay bus driver.

26.2.1.3 $\overline{\text{FR\_A\_TX\_EN}}$ — Transmit Enable Channel A

The $\overline{\text{FR\_A\_TX\_EN}}$ signal indicates to the FlexRay bus driver that the controller is attempting to transmit data on channel A.

26.2.1.4 FR_B_RX — Receive Data Channel B

The FR_B_RX signal carries the receive data for channel B from the corresponding FlexRay bus driver.

26.2.1.5 FR_B_TX — Transmit Data Channel B

The FR_B_TX signal carries the transmit data for channel B to the corresponding FlexRay bus driver

26.2.1.6 $\overline{\text{FR\_B\_TX\_EN}}$ — Transmit Enable Channel B

The $\overline{\text{FR\_B\_TX\_EN}}$ signal indicates to the FlexRay bus driver that the controller is attempting to transmit data on channel B.

26.2.1.7 FR_DBG[3], FR_DBG[2], FR_DBG[1] — , FR_DBG[0] — Strobe Signals

These signals provide the selected debug strobe signals. For details on the debug strobe signal selection refer to Section 26.6.16, Strobe Signal Support.

26.3 Controller Host Interface Clocking

The clock for the CHI is derived from the system bus clock and has the same phase and frequency as the system bus clock. Since the FlexRay protocol requires data delivery at fixed points in time, the memory read cycles from the FlexRay memory must be finished after a fixed amount of time. To ensure this, a minimum frequency f_{chi} of the CHI clock is required, which is given in Equation 26-1.

$$f_{chi} \geq 32\text{MHz} \qquad \textit{Eqn. 26-1}$$

Additional requirements for the minimum frequency of the CHI clock result from the number of message buffers. These requirements are provided in Section 26.7.3, Number of Usable Message Buffers.

NOTE

If a complete message was transmitted from a transmit message buffer or received into a message buffer and the controller host interface (CHI) command FREEZE is issued by the application before the end of the current slot, then this message buffer cannot be disabled and locked until the module has entered the protocol state normal active. Consequently, this message buffer cannot be disabled and locked by the application in the protocol configuration state, which prevents the application from clearing the commit bit CMT and the module from clearing the status bits. The configuration bits in the Message Buffer Configuration, Control, Status Registers (MBCCSRn) and the message buffer configuration registers MBCCFRn, MBFIDRn, and MBIDXRn are not affected. At most one message buffer per channel is affected.

To avoid this,

- The application should not send the CHI command FREEZE and use the CHI command HALT instead.
- Before sending the CHI command FREEZE the application should repeatedly disable all message buffers until all message buffers are disabled. This maximum duration of this task is three static or three dynamic slots.

26.4 Protocol Engine Clocking

The clock for the protocol engine can be generated by two sources. The first source is the internal crystal oscillator and the second source is an internal PLL. The clock source to be used is selected by the clock source select bit CLKSEL in the Module Configuration Register (MCR).

26.4.1 Oscillator Clocking

If the protocol engine is clocked by the internal crystal oscillator, a 40 MHz crystal or CMOS compatible clock must be connected to the oscillator pins. The crystal or clock must fulfill the requirements given by the *FlexRay Communications System Protocol Specification, Version 2.1 Rev A.*

26.4.2 PLL Clocking

26.5 Memory Map and Register Description

The controller occupies 1280 bytes of address space starting at the controller's base address defined by the memory map of the MCU.

26.5.1 Memory Map

The complete memory map of the controller is shown in Table 26-3. The addresses presented here are the offsets relative to the controller base address which is defined by the MCU address map.

Table 26-3. FlexRay Memory Map

Offset	Register	Access
Module Configuration and Control		
0x0000	Module Version Register (MVR)	R
0x0002	Module Configuration Register (MCR)	R/W
0x0004	System Memory Base Address High Register (SYMBADHR)	R/W
0x0006	System Memory Base Address Low Register (SYMBADLR)	R/W
0x0008	Strobe Signal Control Register (STBSCR)	R/W
0x000A	Reserved	R
0x000C	Message Buffer Data Size Register (MBDSR)	R/W
0x000E	Message Buffer Segment Size and Utilization Register (MBSSUTR)	R/W
Test Registers		

Table 26-3. FlexRay Memory Map (continued)

Offset	Register	Access
0x0010	Reserved	R
0x0012	Reserved	R
Interrupt and Error Handling		
0x0014	Protocol Operation Control Register (POCR)	R/W
0x0016	Global Interrupt Flag and Enable Register (GIFER)	R/W
0x0018	Protocol Interrupt Flag Register 0 (PIFR0)	R/W
0x001A	Protocol Interrupt Flag Register 1 (PIFR1)	R/W
0x001C	Protocol Interrupt Enable Register 0 (PIER0)	R/W
0x001E	Protocol Interrupt Enable Register 1 (PIER1)	R/W
0x0020	CHI Error Flag Register (CHIERFR)	R/W
0x0022	Message Buffer Interrupt Vector Register (MBIVEC)	R
0x0024	Channel A Status Error Counter Register (CASERCR)	R
0x0026	Channel B Status Error Counter Register (CBSERCR)	R
Protocol Status		
0x0028	Protocol Status Register 0 (PSR0)	R
0x002A	Protocol Status Register 1 (PSR1)	R
0x002C	Protocol Status Register 2 (PSR2)	R
0x002E	Protocol Status Register 3 (PSR3)	R/W
0x0030	Macrotick Counter Register (MTCTR)	R
0x0032	Cycle Counter Register (CYCTR)	R
0x0034	Slot Counter Channel A Register (SLTCTAR)	R
0x0036	Slot Counter Channel B Register (SLTCTBR)	R
0x0038	Rate Correction Value Register (RTCORVR)	R
0x003A	Offset Correction Value Register (OFCORVR)	R
0x003C	Combined Interrupt Flag Register (CIFRR)	R
0x003E	System Memory Access Time-Out Register (SYMATOR)	R/W
Sync Frame Counter and Tables		
0x0040	Sync Frame Counter Register (SFCNTR)	R
0x0042	Sync Frame Table Offset Register (SFTOR)	R/W
0x0044	Sync Frame Table Configuration, Control, Status Register (SFTCCSR)	R/W
Sync Frame Filter		
0x0046	Sync Frame ID Rejection Filter Register (SFIDRFR)	R/W
0x0048	Sync Frame ID Acceptance Filter Value Register (SFIDAFVR)	R/W
0x004A	Sync Frame ID Acceptance Filter Mask Register (SFIDAFMR)	R/W
Network Management Vector		
0x004C	Network Management Vector Register 0 (NMVR0)	R
0x004E	Network Management Vector Register 1 (NMVR1)	R
0x0050	Network Management Vector Register 2 (NMVR2)	R
0x0052	Network Management Vector Register 3 (NMVR3)	R
0x0054	Network Management Vector Register 4 (NMVR4)	R
0x0056	Network Management Vector Register 5 (NMVR5)	R

Table 26-3. FlexRay Memory Map (continued)

Offset	Register	Access
0x0058	Network Management Vector Length Register (NMVLR)	R/W
Timer Configuration		
0x005A	Timer Configuration and Control Register (TICCR)	R/W
0x005C	Timer 1 Cycle Set Register (TI1CYSR)	R/W
0x005E	Timer 1 Macrotick Offset Register (TI1MTOR)	R/W
0x0060	Timer 2 Configuration Register 0 (TI2CR0)	R/W
0x0062	Timer 2 Configuration Register 1 (TI2CR1)	R/W
Slot Status Configuration		
0x0064	Slot Status Selection Register (SSSR)	R/W
0x0066	Slot Status Counter Condition Register (SSCCR)	R/W
Slot Status		
0x0068	Slot Status Register 0 (SSR0)	R
0x006A	Slot Status Register 1 (SSR1)	R
0x006C	Slot Status Register 2 (SSR2)	R
0x006E	Slot Status Register 3 (SSR3)	R
0x0070	Slot Status Register 4 (SSR4)	R
0x0072	Slot Status Register 5 (SSR5)	R
0x0074	Slot Status Register 6 (SSR6)	R
0x0076	Slot Status Register 7 (SSR7)	R
0x0078	Slot Status Counter Register 0 (SSCR0)	R
0x007A	Slot Status Counter Register 1 (SSCR1)	R
0x007C	Slot Status Counter Register 2 (SSCR2)	R
0x007E	Slot Status Counter Register 3 (SSCR3)	R
MTS Generation		
0x0080	MTS A Configuration Register (MTSACFR)	R/W
0x0082	MTS B Configuration Register (MTSBCFR)	R/W
Shadow Buffer Configuration		
0x0084	Receive Shadow Buffer Index Register (RSBIR)	R/W
Receive FIFO—Configuration		
0x0086	Receive FIFO Watermark and Selection Register (RFWMSR)	R/W
0x0088	Receive FIFO Start Index Register (RFSIR)	R/W
0x008A	Receive FIFO Depth and Size Register (RFDSR)	R/W
Receive FIFO—Control		
0x008C	Receive FIFO A Read Index Register (RFARIR)	R
0x008E	Receive FIFO B Read Index Register (RFBRIR)	R
Receive FIFO—Filter		
0x0090	Receive FIFO Message ID Acceptance Filter Value Register (RFMIDAFVR)	R/W
0x0092	Receive FIFO Message ID Acceptance Filter Mask Register (RFMIAFMR)	R/W
0x0094	Receive FIFO Frame ID Rejection Filter Value Register (RFFIDRFVR)	R/W
0x0096	Receive FIFO Frame ID Rejection Filter Mask Register (RFFIDRFMR)	R/W

Table 26-3. FlexRay Memory Map (continued)

Offset	Register	Access
0x0098	Receive FIFO Range Filter Configuration Register (RFRFCFR)	R/W
0x009A	Receive FIFO Range Filter Control Register (RFRFCTR)	R/W
Dynamic Segment Status		
0x009C	Last Dynamic Transmit Slot Channel A Register (LDTXSLAR)	R
0x009E	Last Dynamic Transmit Slot Channel B Register (LDTXSLBR)	R
Protocol Configuration		
0x00A0 ... 0x00DC	Protocol Configuration Register 0 (PCR0) ... Protocol Configuration Register 30 (PCR30)	R/W – R/W
0x00DE ... 0x00E6	Reserved	R
Receive FIFO—Configuration (continued)		
0x00E8	Receive FIFO System Memory Base Address High Register (RFSYMBADHR)	R/W
0x00EA	Receive FIFO System Memory Base Address Low Register (RFSYMBADLR)	R/W
0x00EC	Receive FIFO Periodic Timer Register (RFPTR)	R/W
Receive FIFO—Control (continued)		
0x00EE	Receive FIFO Fill Level and POP Count Register (RFFLPCR)	R/W
0x00F0 ... 0x00FE	Reserved	R
Message Buffers Configuration, Control, Status		
0x0100	Message Buffer Configuration, Control, Status Register 0 (MBCCSR0)	R/W
0x0102	Message Buffer Cycle Counter Filter Register 0 (MBCCFR0)	R/W
0x0104	Message Buffer Frame ID Register 0 (MBFIDR0)	R/W
0x0106	Message Buffer Index Register 0 (MBIDXR0)	R/W
...	...	...
0x04F8	Message Buffer Configuration, Control, Status Register 127 (MBCCSR127)	R/W
0x04FA	Message Buffer Cycle Counter Filter Register 127 (MBCCFR127)	R/W
0x04FC	Message Buffer Frame ID Register 127 (MBFIDR127)	R/W
0x04FE	Message Buffer Index Register 127 (MBIDXR127)	R/W

26.5.2 Register Descriptions

This section provides detailed descriptions of all registers in ascending address order, presented as 16-bit wide entities

Table 26-4 provides a key for the register figures and register tables.

Table 26-4. Register Access Conventions

Convention	Description
	Depending on its placement in the read or write row, indicates that the bit is not readable or not writeable.
R*	Reserved bit or field, will not be changed. Application must not write any value different from the reset value.
FIELDNAME	Identifies the field. Its presence in the read or write row indicates that it can be read or written.
Register Field Types	
rwm	A read/write bit that may be modified by a hardware in some fashion other than by a reset.
w1c	Write one to clear. A flag bit that can be read, is cleared by writing a one, writing 0 has no effect.
Reset Value	
0	Resets to zero.
1	Resets to one.
–	Not defined after reset and not affected by reset.

26.5.2.1 Register Reset

All registers except the Message Buffer Cycle Counter Filter Registers (MBCCFRn), Message Buffer Frame ID Registers (MBFIDRn), and Message Buffer Index Registers (MBIDXRn) are reset to their reset value on system reset. The registers mentioned above are located in physical memory blocks and, thus, they are not affected by reset. For some register fields, additional reset conditions exist. These additional reset conditions are mentioned in the detailed description of the register. The additional reset conditions are explained in Table 26-5.

Table 26-5. Additional Register Reset Conditions

Condition	Description
Protocol RUN Command	The register field is reset when the application writes to RUN command “0101” to the POCCMD field in the Protocol Operation Control Register (POCR).
Message Buffer Disable	The register field is reset when the application has disabled the message buffer. This happens when the application writes 1 to the message buffer disable trigger bit MBCCSR*n*[EDT] while the message buffer is enabled (MBCCSR*n*[EDS] = 1) and the controller grants the disable to the application by clearing the MBCCSR*n*[EDS] bit.

26.5.2.2 Register Write Access

This section describes the write access restriction terms that apply to all registers.

26.5.2.2.1 Register Write Access Restriction

For each register bit and register field, the write access conditions are specified in the detailed register description. A description of the write access conditions is given in Table 26-6. If, for a specific register bit or field, none of the given write access conditions is fulfilled, any write attempt to this register bit or field is ignored without any notification. The values of the bits or fields are not changed. The condition term [A or B] indicates that the register or field can be written to if at least one of the conditions is fulfilled.

Table 26-6. Register Write Access Restrictions

Condition	Indication	Description
Any Time	—	No write access restriction.
Disabled Mode	MCR[MEN] = 0	Write access only when the controller is in Disabled Mode.
Normal Mode	MCR[MEN] = 1	Write access only when the controller is in Normal Mode.
POC:config	PSR0[PROTSTATE] = *POC:config*	Write access only when the Protocol is in the *POC:config* state.
MB_DIS	MBCCSR*n*[EDS] = 0	Write access only when the related Message Buffer is disabled.
MB_LCK	MBCCSR*n*[LCKS] = 1	Write access only when the related Message Buffer is locked.

26.5.2.2.2 Register Write Access Requirements

All registers can be accessed with 8-bit, 16-bit and 32-bit wide operations. For some of the registers, at least a 16-bit wide write access is required to ensure correct operation. This write access requirement is stated in the detailed register description for each register affected

26.5.2.2.3 Internal Register Access

The following memory mapped registers are used to access multiple internal registers.

- Strobe Signal Control Register (STBSCR)
- Slot Status Selection Register (SSSR)
- Slot Status Counter Condition Register (SSCCR)
- Receive Shadow Buffer Index Register (RSBIR)

Each of these memory mapped registers provides a SEL field and a WMD bit. The SEL field is used to select the internal register. The WMD bit controls the write mode. If the WMD bit is set to 0 during the write access, all fields of the internal register are updated. If the WMD bit set to 1, only the SEL field is changed. All other fields of the internal register remain unchanged. This allows for reading back the values of the selected internal register in a subsequent read access.

26.5.2.3 Module Version Register (MVR)

Base + 0x0000

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CHIVER								PEVER							
W																
Reset	1	0	0	1	1	1	0	1	0	1	1	0	0	1	1	0

Figure 26-2. Module Version Register (MVR)

This register provides the controller version number. The module version number is derived from the CHI version number and the PE version number.

Table 26-7. MVR Field Descriptions

Field	Description
CHIVER	**CHI Version Number**—This field provides the version number of the controller host interface.
PEVER	**PE Version Number**—This field provides the version number of the protocol engine.

26.5.2.4 Module Configuration Register (MCR)

Base + 0x0002 — Write: MEN, SBFF, SCM, CHB, CHA, FUM, FAM, CLKSEL, BITRATE: Disabled Mode
SFFE: Disabled Mode or *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	MEN	SBFF	SCM	CHB	CHA	SFFE	0	0	FUM	FAM	0	CLK SEL	BITRATE			0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-3. Module Configuration Register (MCR)

This register defines the global configuration of the controller.

Table 26-8. MCR Field Descriptions

Field	Description
MEN	**Module Enable** — This bit indicates whether or not the controller is in the Disabled Mode. The application requests the controller to leave the Disabled Mode by writing 1 to this bit Before leaving the Disabled Mode, the application must configure the SCM, SBFF, CHB, CHA, TMODE, BITRATE values. For details see Section 26.1.6, Modes of Operation. 0 Write: ignored, controller disable not possible. Read: controller disabled. 1 Write: enable controller. Read: controller enabled. **Note:** If the controller is enabled it cannot be disabled.
SBFF	**System Bus Failure Freeze** — This bit controls the behavior of the controller in case of a system bus failure. 0 Continue normal operation. 1 Transition to freeze mode.
SCM	**Single Channel Device Mode** — This control bit defines the channel device mode of the controller as described in Section 26.6.10, Channel Device Modes. 0 controller works in dual channel device mode. 1 controller works in single channel device mode.
CHB CHA	**Channel Enable** — protocol related parameter: *pChannels* The semantic of these control bits depends on the channel device mode controlled by the SCM bit and is given Table 26-9.
SFFE	**Synchronization Frame Filter Enable** — This bit controls the filtering for received synchronization frames. For details see Section 26.6.15, Sync Frame Filtering. 0 Synchronization frame filtering disabled. 1 Synchronization frame filtering enabled.
FUM	**FIFO Update Mode** — This bit controls the FIFO update behavior when the interrupt flags GIFER[FAFAIF] and DIFER[FAFBIF] are written by the application (see Section 26.6.9.8, FIFO Update). 0 FIFOA (FIFOB) is updated on writing 1 to GIFER[FAFAIF] (GIFER[FAFBIF]). 1 FIFOA (FIFOB) is *not* updated on writing 1 to GIFER[FAFAIF] (GIFER[FAFBIF]).
FAM	**FIFO Address Mode** — This bit controls the location of the system memory base address for the FIFOs (see Section 26.6.9.2, FIFO Configuration). 0 FIFO Base Address located in System Memory Base Address Register (SYMBADR). 1 FIFO Base Address located in Receive FIFO System Memory Base Address Register (RFSYMBADR).

Table 26-8. MCR Field Descriptions

Field	Description
CLKSEL	**Protocol Engine Clock Source Select** — This bit is used to select the clock source for the protocol engine. 0 PE clock source is generated by on-chip crystal oscillator. 1 PE clock source is generated by on-chip PLL.
BITRATE	**FlexRay Bus Bit Rate** — This bit field defines the FlexRay Bus Bit Rate.00010.0 Mbit/sec 000 10.0 Mbit/sec 001 5.0 Mbit/sec 010 2.5 Mbit/sec 011 8.0 Mbit/sec 100 reserved 101 reserved 110 reserved 111 reserved

Table 26-9. FlexRay Channel Selection

SCM	CHB	CHA	Description
Dual Channel Device Modes			
0	0	0	ports FR_A_RX, FR_A_TX, and $\overline{\text{FR\_A\_TX\_EN}}$ not driven by controller ports FR_B_RX, FR_B_TX, and $\overline{\text{FR\_A\_TX\_EN}}$ not driven by controller
	0	1	ports FR_A_RX, FR_A_TX, and $\overline{\text{FR\_A\_TX\_EN}}$ driven by controller - connected to FlexRay channel A ports FR_B_RX, FR_B_TX, and $\overline{\text{FR\_A\_TX\_EN}}$ not driven by controller
	1	0	ports FR_A_RX, FR_A_TX, and $\overline{\text{FR\_A\_TX\_EN}}$ not driven by controller ports FR_B_RX, FR_B_TX, and $\overline{\text{FR\_A\_TX\_EN}}$ driven by controller - connected to FlexRay channel B
	1	1	ports FR_A_RX, FR_A_TX, and $\overline{\text{FR\_A\_TX\_EN}}$ driven by controller - connected to FlexRay channel A ports FR_B_RX, FR_B_TX, and $\overline{\text{FR\_A\_TX\_EN}}$ driven by controller - connected to FlexRay channel B
Single Channel Device Mode			
1	0	0	ports FR_A_RX, FR_A_TX, and $\overline{\text{FR\_A\_TX\_EN}}$ not driven by controller ports FR_B_RX, FR_B_TX, and $\overline{\text{FR\_A\_TX\_EN}}$ not driven by controller
	0	1	ports FR_A_RX, FR_A_TX, and $\overline{\text{FR\_A\_TX\_EN}}$ driven by controller - connected to FlexRay channel A ports FR_B_RX, FR_B_TX, and $\overline{\text{FR\_A\_TX\_EN}}$ not driven by controller
	1	0	ports FR_A_RX, FR_A_TX, and $\overline{\text{FR\_A\_TX\_EN}}$ driven by controller - connected to FlexRay channel B ports FR_B_RX, FR_B_TX, and $\overline{\text{FR\_A\_TX\_EN}}$ not driven by controller
	1	1	reserved

26.5.2.5 System Memory Base Address Register (SYMBADR)

Base + 0x0004 — Write: Disabled Mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	SMBA[31:16]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-4. System Memory Base Address High Register (SYMBADHR)

Base + 0x0006 Write: Disabled Mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	SMBA[15:4]												0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-5. System Memory Base Address Low Register (SYMBADLR)

NOTE

The system memory base address must be set before the controller is enabled.

The system memory base address registers define the base address of the FlexRay memory within the system memory. The base address is used by the BMIF to calculate the physical memory address for system memory accesses.

Table 26-10. SYMBADR Field Descriptions

Field	Description
SMBA	**System Memory Base Address** — This is the value of the system memory base address for the individual message buffers and sync frame table. This is the value of the system memory base address for the receive FIFO if the FIFO address mode bit MCR[FAM] is set to 1. It is defines as a byte address.

26.5.2.6 Strobe Signal Control Register (STBSCR)

Base + 0x0008 16-bit write access required Write: Anytime

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	SEL				0	0	0	ENB	0	0	STBPSEL	
W	WMD															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-6. Strobe Signal Control Register (STBSCR)

This register is used to assign the individual protocol timing related strobe signals given in Table 26-12 to the external strobe ports. Each strobe signal can be assigned to at most one strobe port. Each write access to registers overwrites the previously written ENB and STBPSEL values for the signal indicated by SEL. If more than one strobe signal is assigned to one strobe port, the current values of the strobe signals are combined with a binary OR and presented at the strobe port. If no strobe signal is assigned to a strobe port, the strobe port carries logic 0. For more detailed and timing information refer to Section 26.6.16, Strobe Signal Support.

NOTE

In single channel device mode, channel B related strobe signals are undefined and should not be assigned to the strobe ports.

Table 26-11. STBSCR Field Descriptions

Field	Description
WMD	**Write Mode** — This control bit defines the write mode of this register. 0 Write to all fields in this register on write access. 1 Write to SEL field only on write access.
SEL	**Strobe Signal Select** — This control field selects one of the strobe signals given in Table 26-12 to be enabled or disabled and assigned to one of the four strobe ports given in Table 26-12.
ENB	**Strobe Signal Enable** — The control bit is used to enable and to disable the strobe signal selected by STBSSEL. 0 Strobe signal is disabled and not assigned to any strobe port. 1 Strobe signal is enabled and assigned to the strobe port selected by STBPSEL.
STBPSEL	**Strobe Port Select** — This field selects the strobe port that the strobe signal selected by the SEL is assigned to. All strobe signals that are enabled and assigned to the same strobe port are combined with a binary OR operation. 00 assign selected signal to FR_DBG[0]. 01 assign selected signal to FR_DBG[1]. 10 assign selected signal to FR_DBG[2]. 11 assign selected signal to FR_DBG[3].

Table 26-12. Strobe Signal Mapping

SEL		Description	Channel	Type	Offset[1]	Reference
dec	hex					
0	0x0	arm	-	value	+1	MT start
1	0x1	mt	-	value	+1	MT start
2	0x2	cycle start	-	pulse	0	MT start
3	0x3	minislot start	-	pulse	0	MT start
4	0x4	slot start	A	pulse	0	MT start
5	0x5		B			
6	0x6	receive data after glitch filtering	A	value	+4	FR_A_RX
7	0x7		B			FR_B_RX
8	0x8	channel idle indicator	A	level	+5	FR_A_RX
9	0x9		B			FR_B_RX
10	0xA	syntax error detected	A	pulse	+4	FR_A_RX
11	0xB		B			FR_B_RX
12	0xC	content error detected	A	level	+4	FR_A_RX
13	0xD		B			FR_B_RX
14	0xE	receive FIFO almost-full interrupt flag	A	value	n.a.	FAFAIF
15	0xF		B			FAFBIF

1 Given in PE clock cycles

26.5.2.7 Message Buffer Data Size Register (MBDSR)

Base + 0x000C — Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	MBSEG2DS							0	MBSEG1DS						
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-7. Message Buffer Data Size Register (MBDSR)

This register defines the size of the message buffer data section for the two message buffer segments in a number of two-byte entities.

The controller provides two independent segments for the individual message buffers. All individual message buffers within one segment have to have the same size for the message buffer data section. This size can be different for the two message buffer segments.

Table 26-13. MBDSR Field Descriptions

Field	Description
MBSEG2DS	**Message Buffer Segment 2 Data Size** — The field defines the size of the message buffer data section in two-byte entities for message buffers within the *second* message buffer segment.
MBSEG1DS	**Message Buffer Segment 1 Data Size** — The field defines the size of the message buffer data section in two-byte entities for message buffers within the *first* message buffer segment.

26.5.2.8 Message Buffer Segment Size and Utilization Register (MBSSUTR)

Base + 0x000E — Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	LAST_MB_SEG1							0	LAST_MB_UTIL						
W																
Reset	0	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1

Figure 26-8. Message Buffer Segment Size and Utilization Register (MBSSUTR)

This register is used to define the last individual message buffer that belongs to the first message buffer segment and the number of the last used individual message buffer.

Table 26-14. MBSSUTR Field Descriptions

Field	Description
LAST_MB_SEG1	**Last Message Buffer In Segment 1** — This field defines the message buffer number of the last individual message buffer that is assigned to the *first* message buffer segment. The individual message buffers in the *first* segment correspond to the message buffer control registers MBCCSR*n*, MBCCFR*n*, MBFIDR*n*, MBIDXR*n* with *n* ≤ LAST_MB_SEG1. The first message buffer segment contains LAST_MB_SEG + 1 individual message buffers. **Note:** The *first* message buffer segment contains *at least* one individual message buffer. The individual message buffers in the *second* message buffer segment correspond to the message buffer control registers MBCCSR*n*, MBCCFR*n*, MBFIDR*n*, MBIDXR*n* with LAST_MB_SEG1 < *n* < 128. **Note:** If LAST_MB_SEG = 127 all individual message buffers belong to the *first* message buffer segment and the *second* message buffer segment is empty.
LAST_MB_UTIL	**Last Message Buffer Utilized** — This field defines the message buffer number of last utilized individual message buffer. The message buffer search engine examines all individual message buffer with a message buffer number *n* ≤ LAST_MB_UTIL. **Note:** If LAST_MB_UTIL = LAST_MB_SEG1 all individual message buffers belong to the *first* message buffer segment and the *second* message buffer segment is empty.

26.5.2.9 Protocol Operation Control Register (POCR)

Base + 0x0014 — Write: Normal Mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	EOC_AP		ERC_AP		BSY	0	0	0	POCCMD			
W	WME								WMC							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-9. Protocol Operation Control Register (POCR)

The application uses this register to issue

- protocol control commands
- external clock correction commands

Protocol control commands are issued by writing to the POCCMD field. For more information on protocol control commands, see Section 26.7.4, Protocol Control Command Execution.

External clock correction commands are issued by writing to the EOC_AP and ERC_AP fields. For more information on external clock correction, refer to Section 26.6.11, External Clock Synchronization.

Table 26-15. POCR Field Descriptions

Field	Description
WME	**Write Mode External Correction** — This bit controls the write mode of the EOC_AP and ERC_AP fields. 0 Write to EOC_AP and ERC_AP fields on register write. 1 No write to EOC_AP and ERC_AP fields on register write.
EOC_AP	**External Offset Correction Application** — This field is used to trigger the application of the external offset correction value defined in the Protocol Configuration Register 29 (PCR29). 00 do not apply external offset correction value 01 reserved 10 Subtract external offset correction value. 11 Add external offset correction value.

Table 26-15. POCR Field Descriptions

Field	Description
ERC_AP	**External Rate Correction Application** — This field is used to trigger application of the external rate correction value defined in the Protocol Configuration Register 21 (PCR21) 00 do not apply external rate correction value 01 reserved 10 Subtract external rate correction value. 11 Add external rate correction value.
BSY	**Protocol Control Command Write Busy** — This status bit indicates the acceptance of the protocol control command issued by the application via the POCCMD field. The controller sets this status bit when the application has issued a protocol control command via the POCCMD field. The controller clears this status bit when protocol control command was accepted by the PE.When the application issues a protocol control command while the BSY bit is asserted, the controller ignores this command, sets the protocol command ignored error flag PCMI_EF in the CHI Error Flag Register (CHIERFR), and will not change the value of the POCCMD field. 0 Command write idle, command accepted and ready to receive new protocol command. 1 Command write busy, command not yet accepted, not ready to receive new protocol command.
WMC	**Write Mode Command** — This bit controls the write mode of the POCCMD field. 0 Write to POCCMD field on register write. 1 Do not write to POCCMD field on register write.
POCCMD	**Protocol Control Command** — The application writes to this field to issue a protocol control command to the PE. The controller sends the protocol command to the PE immediately. While the transfer is running, the BSY bit is set. 0000 ALLOW_COLDSTART — Immediately activate capability of node to cold start cluster. 0001 ALL_SLOTS — Delayed[1] transition to the all slots transmission mode. 0010 CONFIG — Immediately transition to the *POC:config* state. 0011 FREEZE — Immediately transition to the *POC:halt* state. 0100 READY, CONFIG_COMPLETE — Immediately transition to the *POC:ready* state. 0101 RUN — Immediately transition to the *POC:startup start* state. 0110 DEFAULT_CONFIG — Immediately transition to the *POC:default config* state. 0111 HALT — Delayed transition to the *POC:halt* state 1000 WAKEUP — Immediately initiate the wakeup procedure. 1001 reserved 1010 reserved 1011 reserved 1100 reserved 1101 reserved 1110 reserved 1111 reserved

[1] Delayed means on completion of current communication cycle.

26.5.2.10 Global Interrupt Flag and Enable Register (GIFER)

Base + 0x0016 — Write: Normal Mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	MIF	PRIF	CHIF	WUP IF	FAFB IF	FAFA IF	RBIF	TBIF	MIE	PRIE	CHIE	WUP IE	FAFB IE	FAFA IE	RBIE	TBIE
W				w1c	w1c	w1c										
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-10. Global Interrupt Flag and Enable Register (GIFER)

This register provides the means to control some of the interrupt request lines and provides the corresponding interrupt flags. The interrupt flags MIF, PRIF, CHIF, RBIF, and TBIF are the outcome of a binary OR of the related individual interrupt flags and interrupt enables. The generation scheme for these flags is depicted in Figure 26-150. For more details on interrupt generation, see Section 26.6.20, Interrupt Support. These flags are cleared automatically when all of the corresponding interrupt flags or interrupt enables in the related interrupt flag and enable registers are cleared by the application.

Table 26-16. GIFER Field Descriptions

Field	Description
MIF	**Module Interrupt Flag** — This flag is set if at least one of the other interrupt flags is in this register is asserted and the related interrupt enable is asserted, too. The controller generates the module interrupt request if MIE is asserted. 0 No interrupt flag is asserted or no interrupt enable is set. 1 At least one of the other interrupt flags in this register is asserted and the related interrupt bit is asserted, too.
PRIF	**Protocol Interrupt Flag** — This flag is set if at least one of the individual protocol interrupt flags in the Protocol Interrupt Flag Register 0 (PIFR0) and Protocol Interrupt Flag Register 1 (PIFR1) is asserted and the related interrupt enable flag is asserted, too. The controller generates the combined protocol interrupt request if the PRIE flag is asserted. 0 All individual protocol interrupt flags are equal to 0 or no interrupt enable bit is set. 1 At least one of the individual protocol interrupt flags and the related interrupt enable is equal to 1.
CHIF	**CHI Interrupt Flag** — This flag is set if at least one of the individual CHI error flags in the CHI Error Flag Register (CHIERFR) is asserted and the chi error interrupt enable GIFER[CHIE] is asserted. The controller generates the combined CHI error interrupt if the CHIE flag is asserted, too. 0 All CHI error flags are equal to 0 or the chi error interrupt is disabled. 1 At least one CHI error flag is asserted and chi error interrupt is enabled.
WUPIF	**Wakeup Interrupt Flag** — This flag is set when the controller has received a wakeup symbol on the FlexRay bus. The application can determine on which channel the wakeup symbol was received by reading the related wakeup flags WUB and WUA in the Protocol Status Register 3 (PSR3). The controller generates the wakeup interrupt request if the WUPIE flag is asserted. 0 No wakeup condition or interrupt disabled. 1 Wakeup symbol received on FlexRay bus and interrupt enabled.
FAFBIF	**Receive FIFO Channel B Almost Full Interrupt Flag** — This flag is set when one of the following events occurs a) the current number of FIFO B entries is equal to or greater than the watermark defined by the WM field in the Receive FIFO Watermark and Selection Register (RFWMSR), and the controller writes a received message into the FIFO B, or b) the current number of FIFO B entries is at least 1 and the periodic timer as defined by Receive FIFO Periodic Timer Register (RFPTR) expires. 0 No such event. 1 FIFO B almost full event has occurred.
FAFAIF	**Receive FIFO Channel A Almost Full Interrupt Flag** — This flag is set when one of the following events occurs a) the current number of FIFO A entries is equal to or greater than the watermark defined by the WM field in the Receive FIFO Watermark and Selection Register (RFWMSR), and the controller writes a received message into the FIFO A, or b) the current number of FIFO B entries is at least 1 and the periodic timer as defined by Receive FIFO Periodic Timer Register (RFPTR) expires. 0 No such event. 1 FIFO A almost full event has occurred.

Table 26-16. GIFER Field Descriptions (continued)

Field	Description
RBIF	**Receive Message Buffer Interrupt Flag** — This flag is set if for at least one of the individual receive message buffers (MBCCSR*n*[MTD] = 0) both the interrupt flag MBIF and the interrupt enable bit MBIE in the corresponding **Message Buffer Configuration, Control, Status Registers (MBCCSRn)** are asserted. The application cannot clear this RBIF flag directly. This flag is cleared by the controller when all of the interrupt flags MBIF of the individual receive message buffers are cleared by the application or if the application has cleared the interrupt enables bit MBIE. 0 None of the individual receive message buffers has the MBIF and MBIE flag asserted. 1 At least one individual receive message buffer has the MBIF and MBIE flag asserted.
TBIF	**Transmit Buffer Interrupt Flag** — This flag is set if for at least one of the individual single or double transmit message buffers (MBCCSR*n*[MTD] = 0) both the interrupt flag MBIF and the interrupt enable bit MBIE in the corresponding **Message Buffer Configuration, Control, Status Registers (MBCCSRn)** are equal to 1. The application cannot clear this TBIF flag directly. This flag is cleared by the controller when either all of the individual interrupt flags MBIF of the individual transmit message buffers are cleared by the application or the host has cleared the interrupt enables bit MBIE. 0 None of the individual transmit message buffers has the MBIF and MBIE flag asserted. 1 At least one individual transmit message buffer has the MBIF and MBIE flag asserted.
MIE	**Module Interrupt Enable** — This flag controls if the module interrupt line is asserted when the MIF flag is set. 0 Disable interrupt line. 1 Enable interrupt line.
PRIE	**Protocol Interrupt Enable** — This flag controls if the protocol interrupt line is asserted when the PRIF flag is set. 0 Disable interrupt line. 1 Enable interrupt line.
CHIE	**CHI Interrupt Enable** — This flag controls if the CHI interrupt line is asserted when the CHIF flag is set. 0 Disable interrupt line. 1 Enable interrupt line.
WUPIE	**Wakeup Interrupt Enable** — This flag controls if the wakeup interrupt line is asserted when the WUPIF flag is set. 0 Disable interrupt line. 1 Enable interrupt line.
FAFBIE	**Receive FIFO Channel B Almost Full Interrupt Enable** — This flag controls if the FIFO B interrupt line is asserted when the FAFBIF flag is set. 0 Disable interrupt line. 1 Enable interrupt line.
FAFAIE	**Receive FIFO Channel A Almost Full Interrupt Enable** — This flag controls if the FIFO A interrupt line is asserted when the FAFAIF flag is set. 0 Disable interrupt line. 1 Enable interrupt line.
RBIE	**Receive Buffer Interrupt Enable** — This flag controls if the receive buffer interrupt line is asserted when the RBIF flag is set. 0 Disable interrupt line. 1 Enable interrupt line.
TBIE	**Transmit Interrupt Enable** — This flag controls if the transmit buffer interrupt line is asserted when the TBIF flag is set. 0 Disable interrupt line. 1 Enable interrupt line.

26.5.2.11 Protocol Interrupt Flag Register 0 (PIFR0)

Base + 0x0018 — Write: Normal Mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	FATL_IF	INTL_IF	ILCF_IF	CSA_IF	MRC_IF	MOC_IF	CCL_IF	MXS_IF	MTX_IF	LTXB_IF	LTXA_IF	TBVB_IF	TBVA_IF	TI2_IF	TI1_IF	CYS_IF
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-11. Protocol Interrupt Flag Register 0 (PIFR0)

The register holds one set of the protocol-related individual interrupt flags.

Table 26-17. PIFR0 Field Descriptions

Field	Description
FATL_IF	**Fatal Protocol Error Interrupt Flag** — This flag is set when the protocol engine has detected a fatal protocol error. In this case, the protocol engine goes into the *POC:halt* state immediately. The fatal protocol errors are: 1) *pLatestTx* violation, as described in the MAC process of the FlexRay protocol 2) transmission across slot boundary violation, as described in the FSP process of the FlexRay protocol 0 No such event. 1 Fatal protocol error detected.
INTL_IF	**Internal Protocol Error Interrupt Flag** — This flag is set when the protocol engine has detected an internal protocol error. In this case, the protocol engine goes into the *POC:halt* state immediately. An internal protocol error occurs when the protocol engine has not finished a calculation and a new calculation is requested. This can be caused by a hardware error. 0 No such event. 1 Internal protocol error detected.
ILCF_IF	**Illegal Protocol Configuration Interrupt Flag** — This flag is set when the protocol engine has detected an illegal protocol configuration parameter setting. In this case, the protocol engine goes into the *POC:halt* state immediately. The protocol engine checks the *listen_timeout* value programmed into the Protocol Configuration Register 14 (PCR14) and Protocol Configuration Register 15 (PCR15) when the CONFIG_COMPLETE command was sent by the application via the Protocol Operation Control Register (POCR). If the value of *listen_timeout* is equal to zero, the protocol configuration setting is considered as illegal. 0 No such event. 1 Illegal protocol configuration detected.
CSA_IF	**Cold Start Abort Interrupt Flag** — This flag is set when the configured number of allowed cold start attempts is reached and none of these attempts was successful. The number of allowed cold start attempts is configured by the coldstart_attempts field in the Protocol Configuration Register 3 (PCR3). 0 No such event. 1 Cold start aborted and no more coldstart attempts allowed.
MRC_IF	**Missing Rate Correction Interrupt Flag** — This flag is set when an insufficient number of measurements is available for rate correction at the end of the communication cycle. 0 No such event. 1 Insufficient number of measurements for rate correction detected.
MOC_IF	**Missing Offset Correction Interrupt Flag** — This flag is set when an insufficient number of measurements is available for offset correction. This is related to the MISSING_TERM event in the CSP process for offset correction in the FlexRay protocol. 0 No such event. 1 Insufficient number of measurements for offset correction detected.

Table 26-17. PIFR0 Field Descriptions (continued)

Field	Description
CCL_IF	**Clock Correction Limit Reached Interrupt Flag** — This flag is set when the internal calculated offset or rate calculation values have reached or exceeded its configured thresholds as given by the *offset_coorection_out* field in the Protocol Configuration Register 9 (PCR9) and the *rate_correction_out* field in the Protocol Configuration Register 14 (PCR14). 0 No such event. 1 Offset or rate correction limit reached.
MXS_IF	**Max Sync Frames Detected Interrupt Flag** — This flag is set when the number of synchronization frames detected in the current communication cycle exceeds the value of the *node_sync_max* field in the Protocol Configuration Register 30 (PCR30). 0 No such event. 1 More than *node_sync_max* sync frames detected. **Note:** Only synchronization frames that have passed the synchronization frame acceptance and rejection filters are taken into account.
MTX_IF	**Media Access Test Symbol Received Interrupt Flag** — This flag is set when the MTS symbol was received on channel A or channel B. 0 No such event. 1 MTS symbol received.
LTXB_IF	*pLatestTx* **Violation on Channel B Interrupt Flag** — This flag is set when the frame transmission on channel B in the dynamic segment exceeds the dynamic segment boundary. This is related to the *pLatestTx* violation, as described in the MAC process of the FlexRay protocol. 0 No such event. 1 *pLatestTx* violation occurred on channel B.
LTXA_IF	*pLatestTx* **Violation on Channel A Interrupt Flag** — This flag is set when the frame transmission on channel A in the dynamic segment exceeds the dynamic segment boundary. This is related to the *pLatestTx* violation as described in the MAC process of the FlexRay protocol. 0 No such event. 1 *pLatestTx* violation occurred on channel A.
TBVB_IF	**Transmission across boundary on channel B Interrupt Flag** — This flag is set when the frame transmission on channel B crosses the slot boundary. This is related to the transmission across slot boundary violation as described in the FSP process of the FlexRay protocol. 0 No such event. 1 Transmission across boundary violation occurred on channel B.
TBVA_IF	**Transmission across boundary on channel A Interrupt Flag** — This flag is set when the frame transmission on channel A crosses the slot boundary. This is related to the transmission across slot boundary violation as described in the FSP process of the FlexRay protocol. 0 No such event. 1 Transmission across boundary violation occurred on channel A.
TI2_IF	**Timer 2 Expired Interrupt Flag** — This flag is set whenever timer 2 expires. 0 No such event. 1 Timer 2 has reached its time limit.
TI1_IF	**Timer 1 Expired Interrupt Flag** — This flag is set whenever timer 1 expires. 0 No such event 1 Timer 1 has reached its time limit
CYS_IF	**Cycle Start Interrupt Flag** — This flag is set when a communication cycle starts. 0 No such event 1 Communication cycle started.

26.5.2.12 Protocol Interrupt Flag Register 1 (PIFR1)

Base + 0x001A — Write: Normal Mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	EMC_IF	IPC_IF	PECF_IF	PSC_IF	SSI3_IF	SSI2_IF	SSI1_IF	SSI0_IF	0	0	EVT_IF	ODT_IF	0	0	0	0
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c			w1c	w1c				
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-12. Protocol Interrupt Flag Register 1 (PIFR1)

The register holds one set of the protocol-related individual interrupt flags.

Table 26-18. PIFR1 Field Descriptions

Field	Description
EMC_IF	**Error Mode Changed Interrupt Flag** — This flag is set when the value of the ERRMODE bit field in the Protocol Status Register 0 (PSR0) is changed by the controller. 0 No such event. 1 ERRMODE field changed.
IPC_IF	**Illegal Protocol Control Command Interrupt Flag** — This flag is set when the PE tries to execute a protocol control command, which was issued via the POCCMD field of the Protocol Operation Control Register (POCR), and detects that this protocol control command is not allowed in the current protocol state. In this case the command is not executed. For more details, see Section 26.7.4, Protocol Control Command Execution. 0 No such event. 1 Illegal protocol control command detected.
PECF_IF	**Protocol Engine Communication Failure Interrupt Flag** — This flag is set if the controller has detected a communication failure between the protocol engine and the controller host interface 0 No such event. 1 Protocol Engine Communication Failure detected.
PSC_IF	**Protocol State Changed Interrupt Flag** — This flag is set when the protocol state in the PROTSTATE field in the Protocol Status Register 0 (PSR0) has changed. 0 No such event. 1 Protocol state changed.
SSI3_IF SSI2_IF SSI1_IF SSI0_IF	**Slot Status Counter Incremented Interrupt Flag** — Each of these flags is set when the SLOTSTATUSCNT field in the corresponding Slot Status Counter Registers (SSCR0–SSCR3) is incremented. 0 No such event. 1 The corresponding slot status counter has incremented.
EVT_IF	**Even Cycle Table Written Interrupt Flag** — This flag is set if the controller has written the sync frame measurement / ID tables into the FlexRay memory for the even cycle. 0 No such event. 1 Sync frame measurement table written.
ODT_IF	**Odd Cycle Table Written Interrupt Flag** — This flag is set if the controller has written the sync frame measurement / ID tables into the FlexRay memory for the odd cycle. 0 No such event.. 1 Sync frame measurement table written

26.5.2.13 Protocol Interrupt Enable Register 0 (PIER0)

Base + 0x001C — Write: Anytime

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	FATL_IE	INTL_IE	ILCF_IE	CSA_IE	MRC_IE	MOC_IE	CCL_IE	MXS_IE	MTX_IE	LTXB_IE	LTXA_IE	TBVB_IE	TBVA_IE	TI2_IE	TI1_IE	CYS_IE
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-13. Protocol Interrupt Enable Register 0 (PIER0)

This register defines whether or not the individual interrupt flags defined in the Protocol Interrupt Flag Register 0 (PIFR0) can generate a protocol interrupt request.

Table 26-19. PIER0 Field Descriptions

Field	Description
FATL_IE	**Fatal Protocol Error Interrupt Enable** — This bit controls FATL_IF interrupt request generation. 0 interrupt request generation disabled. 1 interrupt request generation enabled.
INTL_IE	**Internal Protocol Error Interrupt Enable** — This bit controls INTL_IF interrupt request generation. 0 interrupt request generation disabled. 1 interrupt request generation enabled.
ILCF_IE	**Illegal Protocol Configuration Interrupt Enable** — This bit controls ILCF_IF interrupt request generation. 0 interrupt request generation disabled. 1 interrupt request generation enabled.
CSA_IE	**Cold Start Abort Interrupt Enable** — This bit controls CSA_IF interrupt request generation. 0 interrupt request generation disabled. 1 interrupt request generation enabled.
MRC_IE	**Missing Rate Correction Interrupt Enable** — This bit controls MRC_IF interrupt request generation. 0 interrupt request generation disabled 1 interrupt request generation enabled.
MOC_IE	**Missing Offset Correction Interrupt Enable** — This bit controls MOC_IF interrupt request generation. 0 interrupt request generation disabled. 1 interrupt request generation enabled.
CCL_IE	**Clock Correction Limit Reached Interrupt Enable** — This bit controls CCL_IF interrupt request generation. 0 interrupt request generation disabled. 1 interrupt request generation enabled.
MXS_IE	**Max Sync Frames Detected Interrupt Enable** — This bit controls MXS_IF interrupt request generation. 0 interrupt request generation disabled. 1 interrupt request generation enabled.
MTX_IE	**Media Access Test Symbol Received Interrupt Enable** — This bit controls MTX_IF interrupt request generation. 0 interrupt request generation disabled. 1 interrupt request generation enabled.
LTXB_IE	*pLatestTx* **Violation on Channel B Interrupt Enable** — This bit controls LTXB_IF interrupt request generation. 0 interrupt request generation disabled. 1 interrupt request generation enabled.

Table 26-19. PIER0 Field Descriptions (continued)

Field	Description
LTXA_IE	*pLatestTx* **Violation on Channel A Interrupt Enable** — This bit controls LTXA_IF interrupt request generation. 0 interrupt request generation disabled. 1 interrupt request generation enabled.
TBVB_IE	**Transmission across boundary on channel B Interrupt Enable** — This bit controls TBVB_IF interrupt request generation. 0 interrupt request generation disabled. 1 interrupt request generation enabled.
TBVA_IE	**Transmission across boundary on channel A Interrupt Enable** — This bit controls TBVA_IF interrupt request generation. 0 interrupt request generation disabled. 1 interrupt request generation enabled.
TI2_IE	**Timer 2 Expired Interrupt Enable** — This bit controls TI1_IF interrupt request generation. 0 interrupt request generation disabled. 1 interrupt request generation enabled.
TI1_IE	**Timer 1 Expired Interrupt Enable** — This bit controls TI1_IF interrupt request generation. 0 interrupt request generation disabled. 1 interrupt request generation enabled.
CYS_IE	**Cycle Start Interrupt Enable** — This bit controls CYC_IF interrupt request generation. 0 interrupt request generation disabled. 1 interrupt request generation enabled.

26.5.2.14 Protocol Interrupt Enable Register 1 (PIER1)

Base + 0x001E — Write: Anytime

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	EMC_IE	IPC_IE	PECF_IE	PSC_IE	SSI3_IE	SSI2_IE	SSI1_IE	SSI0_IE	0	0	EVT_IE	ODT_IE	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-14. Protocol Interrupt Enable Register 1 (PIER1)

This register defines whether or not the individual interrupt flags defined in Protocol Interrupt Flag Register 1 (PIFR1) can generate a protocol interrupt request.

Table 26-20. PIER1 Field Descriptions

Field	Description
EMC_IE	**Error Mode Changed Interrupt Enable** — This bit controls EMC_IF interrupt request generation. 0 interrupt request generation disabled. 1 interrupt request generation enabled.
IPC_IE	**Illegal Protocol Control Command Interrupt Enable** — This bit controls IPC_IF interrupt request generation. 0 interrupt request generation disabled. 1 interrupt request generation enabled.

Table 26-20. PIER1 Field Descriptions (continued)

Field	Description
PECF_IE	**Protocol Engine Communication Failure Interrupt Enable** — This bit controls PECF_IF interrupt request generation. 0 interrupt request generation disabled. 1 interrupt request generation enabled.
PSC_IE	**Protocol State Changed Interrupt Enable** — This bit controls PSC_IF interrupt request generation. 0 interrupt request generation disabled. 1 interrupt request generation enabled.
SSI3_IE SSI2_IE SSI1_IE SSI0_IE	**Slot Status Counter Incremented Interrupt Enable** — This bit controls SSI[3:0]_IF interrupt request generation. 0 interrupt request generation disabled. 1 interrupt request generation enabled.
EVT_IE	**Even Cycle Table Written Interrupt Enable** — This bit controls EVT_IF interrupt request generation. 0 interrupt request generation disabled. 1 interrupt request generation enabled.
ODT_IE	**Odd Cycle Table Written Interrupt Enable** — This bit controls ODT_IF interrupt request generation. 0 interrupt request generation disabled. 1 interrupt request generation enabled.

26.5.2.15 CHI Error Flag Register (CHIERFR)

Base + 0x0020 — Write: Normal Mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	FRLB_EF	FRLA_EF	PCMI_EF	FOVB_EF	FOVA_EF	MBS_EF	MBU_EF	LCK_EF	DBL_EF	SBCF_EF	FID_EF	DPL_EF	SPL_EF	NML_EF	NMF_EF	ILSA_EF
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-15. CHI Error Flag Register (CHIERFR)

This register holds the CHI related error flags. The interrupt generation for each of these error flags is controlled by the CHI interrupt enable bit CHIE in the Global Interrupt Flag and Enable Register (GIFER).

Table 26-21. CHIERFR Field Descriptions

Field	Description
FRLB_EF	**Frame Lost Channel B Error Flag** — This flag is set if a complete frame was received on channel B but could not be stored in the selected individual message buffer because this message buffer is currently locked by the application. In this case, the frame and the related slot status information are lost. 0 No such event. 1 Frame lost on channel B detected.
FRLA_EF	**Frame Lost Channel A Error Flag** — This flag is set if a complete frame was received on channel A but could not be stored in the selected individual message buffer because this message buffer is currently locked by the application. In this case, the frame and the related slot status information are lost. 0 No such error. 1 Frame lost on channel A detected.

Table 26-21. CHIERFR Field Descriptions (continued)

Field	Description
PCMI_EF	**Protocol Command Ignored Error Flag** — This flag is set if the application has issued a POC command by writing to the POCCMD field in the Protocol Operation Control Register (POCR) while the BSY flag is equal to 1. In this case the command is ignored by the controller and is lost. 0 No such error. 1 POC command ignored.
FOVB_EF	**Receive FIFO Overrun Channel B Error Flag** — This flag is set when an overrun of the FIFO for channel B occurred. This error occurs if a semantically valid frame was received on channel B and matches the all criteria to be appended to the FIFO for channel B but the FIFO is full. In this case, the received frame and its related slot status information is lost. 0 No such error. 1 FIFO overrun on channel B has been detected.
FOVA_EF	**Receive FIFO Overrun Channel A Error Flag** — This flag is set when an overrun of the FIFO for channel A occurred. This error occurs if a semantically valid frame was received on channel A and matches the all criteria to be appended to the FIFO for channel A but the FIFO is full. In this case, the received frame and its related slot status information is lost. 0 No such error. 1 FIFO overrun on channel B has been detected.
MSB_EF	**Message Buffer Search Error Flag** — This flag is set if the message buffer search engine is still running while the next search cycle must be started due to the FlexRay protocol timing. In this case, not all message buffers are considered while searching. 0 No such event. 1 Search engine active while search start appears.
MBU_EF	**Message Buffer Utilization Error Flag** — This flag is asserted if the application writes to a message buffer control field that is beyond the number of utilized message buffers programmed in the Message Buffer Segment Size and Utilization Register (MBSSUTR). If the application writes to a MBCCSR*n* register with *n* > LAST_MB_UTIL, the controller ignores the write attempt and asserts the message buffer utilization error flag MBU_EF in the CHI Error Flag Register (CHIERFR). 0 No such event. 1 Non-utilized message buffer enabled.
LCK_EF	**Lock Error Flag** — This flag is set if the application tries to lock a message buffer that is already locked by the controller due to internal operations. In that case, the controller does not grant the lock to the application. The application must issue the lock request again. 0 No such error. 1 Lock error detected.
DBL_EF	**Double Transmit Message Buffer Lock Error Flag** — This flag is set if the application tries to lock the transmit side of a double transmit message buffer. In this case, the controller does not grant the lock to the transmit side of a double transmit message buffer. 0 No such event. 1 Double transmit buffer lock error occurred.
SBCF_EF	**System Bus Communication Failure Error Flag** — This flag is set if a system bus access was not finished within the required amount of time (see Section 26.6.19.2, System Bus Access Timeout). 0 No such event. 1 System bus access not finished in time.
FID_EF	**Frame ID Error Flag** — This flag is set if the frame ID stored in the message buffer header area differs from the frame ID stored in the message buffer control register. 0 No such error occurred. 1 Frame ID error occurred.

Table 26-21. CHIERFR Field Descriptions (continued)

Field	Description
DPL_EF	**Dynamic Payload Length Error Flag** — This flag is set if the payload length written into the message buffer header field of a single or double transmit message buffer assigned to the dynamic segment is greater than the maximum payload length for the dynamic segment as it is configured in the corresponding protocol configuration register field max_payload_length_dynamic in the Protocol Configuration Register 24 (PCR24). 0 No such error occurred. 1 Dynamic payload length error occurred.
SPL_EF	**Static Payload Length Error** Flag — This flag is set if the payload length written into the message buffer header field of a single or double transmit message buffer assigned to the static segment is different from the payload length for the static segment as it is configured in the corresponding protocol configuration register field payload_length_static in the Protocol Configuration Register 19 (PCR19). 0 No such error occurred. 1 Static payload length error occurred.
NML_EF	**Network Management Length Error Flag** — This flag is set if the payload length written into the header structure of a receive message buffer assigned to the static segment is less than the configured length of the Network Management Vector as configured in the Network Management Vector Length Register (NMVLR). In this case the received part of the Network Management Vector will be used to update the Network Management Vector. 0 No such error occurred. 1 Network management length error occurred.
NMF_EF	**Network Management Frame Error Flag** — This flag is set if a received message in the static segment with a Preamble Indicator flag PP asserted has its Null Frame indicator flag NF asserted as well. In this case, the Global Network Management Registers (see Network Management Vector Registers (NMVR0–NMVR5)) are not updated. 0 No such error occurred. 1 Network management frame error occurred.
ILSA_EF	**Illegal System Bus Address Error Flag** — This flag is set if the external system bus subsystem has detected an access to an illegal system bus address from the controller (see Section 26.6.19.1, System Bus Illegal Address Access). 0 No such event. 1 Illegal system bus address accessed.

26.5.2.16 Message Buffer Interrupt Vector Register (MBIVEC)

Base + 0x0022

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	TBIVEC							0	RBIVEC						
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-16. Message Buffer Interrupt Vector Register (MBIVEC)

This register indicates the lowest numbered receive message buffer and the lowest numbered transmit message buffer that have their interrupt status flag MBIF and interrupt enable MBIE bits asserted. This means that message buffers with lower message buffer numbers have higher priority.

Table 26-22. MBIVEC Field Descriptions

Field	Description
TBIVEC	**Transmit Buffer Interrupt Vector** — This field provides the number of the lowest numbered enabled transmit message buffer that has its interrupt status flag MBIF and its interrupt enable bit MBIE set. If there is no transmit message buffer with the interrupt status flag MBIF and the interrupt enable MBIE bits asserted, the value in this field is set to 0.
RBIVEC	**Receive Buffer Interrupt Vector** — This field provides the message buffer number of the lowest numbered receive message buffer which has its interrupt flag MBIF and its interrupt enable bit MBIE asserted. If there is no receive message buffer with the interrupt status flag MBIF and the interrupt enable MBIE bits asserted, the value in this field is set to 0.

26.5.2.17 Channel A Status Error Counter Register (CASERCR)

Base + 0x0024 — Additional Reset: RUN Command

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	STATUS_ERR_CNT															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-17. Channel A Status Error Counter Register (CASERCR)

This register provides the channel status error counter for channel A. The protocol engine generates a slot status vector for each static slot, each dynamic slot, the symbol window, and the NIT. The slot status vector contains the four protocol related error indicator bits *vSS!SyntaxError, vSS!ContentError, vSS!BViolation*, and *vSS!TxConflict*. The controller increments the status error counter by 1 if, for a slot or segment, at least one error indicator bit is set to 1. The counter wraps around after it has reached the maximum value. For more information on slot status monitoring, see Section 26.6.18, Slot Status Monitoring.

Table 26-23. CASERCR Field Descriptions

Field	Description
STATUS_ERR_CNT	**Channel Status Error Counter** — This field provides the current value channel status error counter. The counter value is updated within the first macrotick of the following slot or segment.

26.5.2.18 Channel B Status Error Counter Register (CBSERCR)

Base + 0x0026 — Additional Reset: RUN Command

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	STATUS_ERR_CNT															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-18. Channel B Status Error Counter Register (CBSERCR)

This register provides the channel status error counter for channel B. The protocol engine generates a slot status vector for each static slot, each dynamic slot, the symbol window, and the NIT. The slot status vector contains the four protocol related error indicator bits *vSS!SyntaxError*, *vSS!ContentError*, *vSS!BViolation*, and *vSS!TxConflict*. The controller increments the status error counter by 1 if, for a slot or segment, at least

one error indicator bit is set to 1. The counter wraps around after it has reached the maximum value. For more information on slot status monitoring see Section 26.6.18, Slot Status Monitoring.

Table 26-24. CBSERCR Field Descriptions

Field	Description
STATUS_ERR_CNT	**Channel Status Error Counter** — This field provides the current channel status error count. The counter value is updated within the first macrotick of the following slot or segment.

26.5.2.19 Protocol Status Register 0 (PSR0)

Base + 0x0028

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	ERRMODE		SLOTMODE		0	PROTSTATE			STARTUPSTATE				0	WAKEUPSTATUS		
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-19. Protocol Status Register 0 (PSR0)

This register provides information about the current protocol status.

Table 26-25. PSR0 Field Descriptions

Field	Description
ERRMODE	**Error Mode** — protocol related variable: *vPOC!ErrorMode*. This field indicates the error mode of the protocol. 00 ACTIVE 01 PASSIVE 10 COMM_HALT 11 reserved
SLOTMODE	**Slot Mode** — protocol related variable: *vPOC!SlotMode*. This field indicates the slot mode of the protocol. 00 SINGLE 01 ALL_PENDING 10 ALL 11 reserved
PROTSTATE	**Protocol State** — protocol related variable: *vPOC!State.* This field indicates the state of the protocol. 000 *POC:default config* 001 *POC:config* 010 *POC:wakeup* 011 *POC:ready* 100 *POC:normal passive* 101 *POC:normal active* 110 *POC:halt* 111 *POC:startup*

Table 26-25. PSR0 Field Descriptions (continued)

Field	Description
STARTUP STATE	**Startup State** — protocol related variable: *vPOC!StartupState.* This field indicates the current sub-state of the startup procedure. 0000 reserved 0001 reserved 0010 *POC:coldstart collision resolution* 0011 *POC:coldstart listen* 0100 *POC:integration consistency check* 0101 *POC:integrationi listen* 0110 reserved 0111 *POC:initialize schedule* 1000 reserved 1001 reserved 1010 *POC:coldstart consistency check* 1011 reserved 1100 reserved 1101 *POC:integration coldstart check* 1110 *POC:coldstart gap* 1111 *POC:coldstart join*
WAKEUP STATUS	**Wakeup Status** — protocol related variable: *vPOC!WakeupStatus*. This field provides the outcome of the execution of the wakeup mechanism. 000 UNDEFINED 001 RECEIVED_HEADER 010 RECEIVED_WUP 011 COLLISION_HEADER 100 COLLISION_WUP 101 COLLISION_UNKNOWN 110 TRANSMITTED 111 reserved

26.5.2.20 Protocol Status Register 1 (PSR1)

Base + 0x002A — Additional Reset: CSAA, CSP, CPN: RUN Command — Write: Normal Mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CSAA	CSP	0	REMCSAT					CPN	HHR	FRZ	APTAC				
W	w1c															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-20. Protocol Status Register 1 (PSR1)

Table 26-26. PSR1 Field Descriptions

Field	Description
CSAA	**Cold Start Attempt Aborted Flag** — protocol related event: ‘set coldstart abort indicator in CHI’ This flag is set when the controller has aborted a cold start attempt. 0 No such event. 1 Cold start attempt aborted.
CSP	**Leading Cold Start Path** — This status bit is set when the controller has reached the *POC:normal active* state via the leading cold start path. This indicates that this node has started the network 0 No such event. 1 *POC:normal active* reached from *POC:startup* state via leading cold start path.

Table 26-26. PSR1 Field Descriptions (continued)

Field	Description
REMCSAT	**Remaining Coldstart Attempts** — protocol related variable: *vRemainingColdstartAttempts* This field provides the number of remaining cold start attempts that the controller will execute.
CPN	**Leading Cold Start Path Noise** — protocol related variable: *vPOC!ColdstartNoise* This status bit is set if the controller has reached the *POC:normal active* state via the leading cold start path under noise conditions. This indicates there was some activity on the FlexRay bus while the controller was starting up the cluster. 0 No such event. 1 *POC:normal active* state was reached from *POC:startup* state via noisy leading cold start path.
HHR	**Host Halt Request Pending** — protocol related variable: *vPOC!CHIHaltRequest* This status bit is set when controller receives the HALT command from the application via the Protocol Operation Control Register (POCR). The controller clears this status bit after a hard reset condition or when the protocol is in the *POC:default config* state. 0 No such event. 1 HALT command received.
FRZ	**Freeze Occurred** — protocol related variable: *vPOC!Freeze* This status bit is set when the controller has reached the *POC:halt* state due to the host FREEZE command or due to an internal error condition requiring immediate halt. The controller clears this status bit after a hard reset condition or when the protocol is in the *POC:default config* state. 0 No such event. 1 Immediate halt due to FREEZE or internal error condition.
APTAC	**Allow Passive to Active Counter** — protocol related variable: *vPOC!vAllowPassivetoActive* This field provides the number of consecutive even/odd communication cycle pairs that have passed with valid rate and offset correction terms, but the protocol is still in the *POC:normal passive* state due to an application configured delay to enter *POC:normal active* state. This delay is defined by the allow_passive_to_active field in the Protocol Configuration Register 12 (PCR12).

26.5.2.21 Protocol Status Register 2 (PSR2)

Base + 0x002C — Additional Reset: RUN Command

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	NBVB	NSEB	STCB	SBVB	SSEB	MTB	NBVA	NSEA	STCA	SBVA	SSEA	MTA	CLKCORRFAILCNT			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-21. Protocol Status Register 2 (PSR2)

This register provides a snapshot of status information about the Network Idle Time NIT, the Symbol Window and the clock synchronization. The NIT related status bits NBVB, NSEB, NBVA, and NSEA are updated by the controller after the end of the NIT and before the end of the first slot of the next communication cycle. The Symbol Window related status bits STCB, SBVB, SSEB, MTB, STCA, SBVA, SSEB, and MTA are updated by the controller after the end of the symbol window and before the end of the current communication cycle. If no symbol window is configured, the symbol window related status bits remain in their reset state. The clock synchronization related CLKCORRFAILCNT is updated by the controller after the end of the static segment and before the end of the current communication cycle.

Table 26-27. PSR2 Field Descriptions

Field	Description
NBVB	**NIT Boundary Violation on Channel B** — protocol related variable: *vSS!BViolation* for NIT on channel B. This status bit is set when there was some media activity on the FlexRay bus channel B at the end of the NIT. 0 No such event. 1 Media activity at boundaries detected.
NSEB	**NIT Syntax Error on Channel B** — protocol related variable: *vSS!SyntaxError* for NIT on channel B. This status bit is set when a syntax error was detected during NIT on channel B. 0 No such event. 1 Syntax error detected.
STCB	**Symbol Window Transmit Conflict on Channel B** — protocol related variable: *vSS!TxConflict* for symbol window on channel B. This status bit is set if there was a transmission conflict during the symbol window on channel B. 0 No such event. 1 Transmission conflict detected.
SBVB	**Symbol Window Boundary Violation on Channel B** — protocol related variable: *vSS!BViolation* for symbol window on channel B. This status bit is set if there was some media activity on the FlexRay bus channel B at the start or at the end of the symbol window. 0 No such event. 1 Media activity at boundaries detected.
SSEB	**Symbol Window Syntax Error on Channel B** — protocol related variable: *vSS!SyntaxError* for symbol window on channel B. This status bit is set when a syntax error was detected during the symbol window on channel B. 0 No such event. 1 Syntax error detected.
MTB	**Media Access Test Symbol MTS Received on Channel B** — protocol related variable: *vSS!ValidMTS* for Symbol Window on channel B. This status bit is set if the Media Access Test Symbol MTS was received in the symbol window on channel B. 0 No such event. 1 MTS symbol received.
NBVA	**NIT Boundary Violation on Channel A** — protocol related variable: *vSS!BViolation* for NIT on channel A This status bit is set when there was some media activity on the FlexRay bus channel A at the end of the NIT. 0 No such event. 1 Media activity at boundaries detected.
NSEA	**NIT Syntax Error on Channel A** — protocol related variable: *vSS!SyntaxError* for NIT on channel A. This status bit is set when a syntax error was detected during NIT on channel A. 0 No such event. 1 Syntax error detected.
STCA	**Symbol Window Transmit Conflict on Channel A** — protocol related variable: *vSS!TxConflict* for symbol window on channel A. This status bit is set if there was a transmission conflicts during the symbol window on channel A. 0 No such event. 1 Transmission conflict detected.
SBVA	**Symbol Window Boundary Violation on Channel A** — protocol related variable: *vSS!BViolation* for symbol window on channel A. This status bit is set if there was some media activity on the FlexRay bus channel A at the start or at the end of the symbol window. 0 No such event. 1 Media activity at boundaries detected.

Table 26-27. PSR2 Field Descriptions (continued)

Field	Description
SSEA	**Symbol Window Syntax Error on Channel A** — protocol related variable: *vSS!SyntaxError* for symbol window on channel A. This status bit is set when a syntax error was detected during the symbol window on channel A. 0 No such event. 1 Syntax error detected.
MTA	**Media Access Test Symbol MTS Received on Channel A** — protocol related variable: *vSS!ValidMTS* for symbol window on channel A. This status bit is set if the Media Access Test Symbol MTS was received in the symbol window on channel A. 1 MTS symbol received. 0 No such event.
CLKCORR-FAILCNT	**Clock Correction Failed Counter** — protocol related variable: *vClockCorrectionFailed* This field provides the number of consecutive even/odd communication cycle pairs that have passed without clock synchronization having performed an offset or a rate correction due to lack of synchronization frames. It is not incremented when it has reached the configured value of either max_without_clock_correction_fatal or max_without_clock_correction_passive as defined in the Protocol Configuration Register 8 (PCR8). The controller resets this counter on a hard reset condition, when the protocol enters the *POC:normal active* state, or when both the rate and offset correction terms have been calculated successfully.

26.5.2.22 Protocol Status Register 3 (PSR3)

Base + 0x002E — Additional Reset: RUN Command — Write: Normal Mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	WUB	ABVB	AACB	ACEB	ASEB	AVFB	0	0	WUA	ABVA	AACA	ACEA	ASEA	AVFA
W			w1c	w1c	w1c	w1c	w1c	w1c			w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-22. Protocol Status Register 3 (PSR3)

This register provides aggregated channel status information as an accrued status of channel activity for all communication slots, regardless of whether they are assigned for transmission or subscribed for reception. It provides accrued information for the symbol window, the NIT, and the wakeup status.

Table 26-28. PSR3 Field Descriptions

Field	Description
WUB	**Wakeup Symbol Received on Channel B** — This flag is set when a wakeup symbol was received on channel B. 0 No wakeup symbol received. 1 Wakeup symbol received.
ABVB	**Aggregated Boundary Violation on Channel B** — This flag is set when a boundary violation has been detected on channel B. Boundary violations are detected in the communication slots, the symbol window, and the NIT. 0 No boundary violation detected. 1 Boundary violation detected.
AACB	**Aggregated Additional Communication on Channel B** — This flag is set when at least one valid frame was received on channel B in a slot that also contained an additional communication with either syntax error, content error, or boundary violations. 0 No additional communication detected. 1 Additional communication detected.

Table 26-28. PSR3 Field Descriptions (continued)

Field	Description
ACEB	**Aggregated Content Error on Channel B** — This flag is set when a content error has been detected on channel B. Content errors are detected in the communication slots, the symbol window, and the NIT. 0 No content error detected. 1 Content error detected.
ASEB	**Aggregated Syntax Error on Channel B** — This flag is set when a syntax error has been detected on channel B. Syntax errors are detected in the communication slots, the symbol window and the NIT. 0 No syntax error detected. 1 Syntax errors detected.
AVFB	**Aggregated Valid Frame on Channel B** — This flag is set when a syntactically correct valid frame has been received in any static or dynamic slot through channel B. 1 At least one syntactically valid frame received. 0 No syntactically valid frames received.
WUA	**Wakeup Symbol Received on Channel A** — This flag is set when a wakeup symbol was received on channel A. 0 No wakeup symbol received. 1 Wakeup symbol received.
ABVA	**Aggregated Boundary Violation on Channel A** — This flag is set when a boundary violation has been detected on channel A. Boundary violations are detected in the communication slots, the symbol window, and the NIT. 0 No boundary violation detected. 1 Boundary violation detected.
AACA	**Aggregated Additional Communication on Channel A** — This flag is set when a valid frame was received in a slot on channel A that also contained an additional communication with either syntax error, content error, or boundary violations. 0 No additional communication detected. 1 Additional communication detected.
ACEA	**Aggregated Content Error on Channel A** — This flag is set when a content error has been detected on channel A. Content errors are detected in the communication slots, the symbol window, and the NIT. 0 No content error detected. 1 Content error detected.
ASEA	**Aggregated Syntax Error on Channel A** — This flag is set when a syntax error has been detected on channel A. Syntax errors are detected in the communication slots, the symbol window, and the NIT. 0 No syntax error detected. 1 Syntax errors detected.
AVFA	**Aggregated Valid Frame on Channel A** — This flag is set when a syntactically correct valid frame has been received in any static or dynamic slot through channel A. 0 No syntactically valid frames received. 1 At least one syntactically valid frame received.

26.5.2.23 Macrotick Counter Register (MTCTR)

Base + 0x0030

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	MTCT													
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-23. Macrotick Counter Register (MTCTR)

This register provides the macrotick count of the current communication cycle.

Table 26-29. MTCTR Field Descriptions

Field	Description
MTCT	**Macrotick Counter** — protocol related variable: *vMacrotick* This field provides the macrotick count of the current communication cycle.

26.5.2.24 Cycle Counter Register (CYCTR)

Base + 0x0032

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	CYCCNT					
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-24. Cycle Counter Register (CYCTR)

This register provides the number of the current communication cycle.

Table 26-30. CYCTR Field Descriptions

Field	Description
CYCCNT	**Cycle Counter** — protocol related variable: *vCycleCounter* This field provides the number of the current communication cycle. If the counter reaches the maximum value of 63, the counter wraps and starts from zero again.

26.5.2.25 Slot Counter Channel A Register (SLTCTAR)

Base + 0x0034

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	SLOTCNTA										
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-25. Slot Counter Channel A Register (SLTCTAR)

This register provides the number of the current slot in the current communication cycle for channel A.

Table 26-31. SLTCTAR Field Descriptions

Field	Description
SLOTCNTA	**Slot Counter Value for Channel A** — protocol related variable: *vSlotCounter* for channel A This field provides the number of the current slot in the current communication cycle.

26.5.2.26 Slot Counter Channel B Register (SLTCTBR)

Base + 0x0036

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	SLOTCNTB										
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-26. Slot Counter Channel B Register (SLTCTBR)

This register provides the number of the current slot in the current communication cycle for channel B.

Table 26-32. SLTCTBR Field Descriptions

Field	Description
SLOTCNTA	**Slot Counter Value for Channel B** — protocol related variable: *vSlotCounter* for channel B This field provides the number of the current slot in the current communication cycle.

26.5.2.27 Rate Correction Value Register (RTCORVR)

Base + 0x0038 Additional Reset: RUN Command

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	RATECORR															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-27. Rate Correction Value Register (RTCORVR)

This register provides the sign extended rate correction value in microticks as it was calculated by the clock synchronization algorithm. The controller updates this register during the NIT of each odd numbered communication cycle.

Table 26-33. RTCORVR Field Descriptions

Field	Description
RATECORR	**Rate Correction Value** — protocol related variable: *vRateCorrection* (before value limitation and external rate correction). This field provides the sign extended rate correction value in microticks as it was calculated by the clock synchronization algorithm. The value is represented in 2's complement format. This value does not include the value limitation and the application of the external rate correction. If the magnitude of the internally calculated rate correction value exceeds the limit given by rate_correction_out in the Protocol Configuration Register 13 (PCR13), the clock correction reached limit interrupt flag CCL_IF is set in the Protocol Interrupt Flag Register 0 (PIFR0). **Note:** If the controller was not able to calculate a new rate correction term due to a lack of synchronization frames, the RATECORR value is not updated.

26.5.2.28 Offset Correction Value Register (OFCORVR)

Base + 0x003A — Additional Reset: RUN Command

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	OFFSETCORR															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-28. Offset Correction Value Register (OFCORVR)

This register provides the sign extended offset correction value in microticks as it was calculated by the clock synchronization algorithm. The controller updates this register during the NIT.

Table 26-34. OFCORVR Field Descriptions

Field	Description
OFFSET CORR	**Offset Correction Value** — protocol related variable: *vOffsetCorrection* (before value limitation and external offset correction). This field provides the sign extended offset correction value in microticks as it was calculated by the clock synchronization algorithm. The value is represented in 2's complement format. This value does not include the value limitation and the application of the external offset correction. If the magnitude of the internally calculated rate correction value exceeds the limit given by offset_correction_out field in the Protocol Configuration Register 29 (PCR29), the clock correction reached limit interrupt flag CCL_IF is set in the Protocol Interrupt Flag Register 0 (PIFR0). **Note:** If the controller was not able to calculate an new offset correction term due to a lack of synchronization frames, the OFFSETCORR value is not updated.

26.5.2.29 Combined Interrupt Flag Register (CIFRR)

Base + 0x003C

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	MIF	PRIF	CHIF	WUP IF	FAFB IF	FAFA IF	RBIF	TBIF
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-29. Combined Interrupt Flag Register (CIFRR)

This register provides five combined interrupt flags and a copy of three individual interrupt flags. The combined interrupt flags are the result of a binary OR of the values of other interrupt flags regardless of the state of the interrupt enable bits. The generation scheme for the combined interrupt flags is depicted in Figure 26-152. The individual interrupt flags WUPIF, FAFBIF, and FAFAIF are copies of corresponding flags in the Global Interrupt Flag and Enable Register (GIFER) and are provided here to simplify the application interrupt flag check. To clear the individual interrupt flags, the application must use the Global Interrupt Flag and Enable Register (GIFER).

NOTE

The meanings of the combined status bits MIF, PRIF, CHIF, RBIF, and TBIF are different from those mentioned in the Global Interrupt Flag and Enable Register (GIFER).

Table 26-35. CIFRR Field Descriptions

Field	Description
MIF	**Module Interrupt Flag** — This flag is set if there is at least one interrupt source that has its interrupt flag asserted. 0 No interrupt source has its interrupt flag asserted. 1 At least one interrupt source has its interrupt flag asserted.
PRIF	**Protocol Interrupt Flag** — This flag is set if at least one of the individual protocol interrupt flags in the Protocol Interrupt Flag Register 0 (PIFR0) or Protocol Interrupt Flag Register 1 (PIFR1) is equal to 1. 0 All individual protocol interrupt flags are equal to 0. 1 At least one of the individual protocol interrupt flags is equal to 1.
CHIF	**CHI Interrupt Flag** — This flag is set if at least one of the individual CHI error flags in the CHI Error Flag Register (CHIERFR) is equal to 1. 0 All CHI error flags are equal to 0. 1 At least one CHI error flag is equal to 1.
WUPIF	**Wakeup Interrupt Flag** — Provides the same value as GIFER[WUPIF].
FAFBIF	**Receive FIFO Channel B Almost Full Interrupt Flag** — Provides the same value as GIFER[FAFBIF].
FAFAIF	**Receive FIFO Channel A Almost Full Interrupt Flag** — Provides the same value as GIFER[FAFAIF].
RBIF	**Receive Message Buffer Interrupt Flag** — This flag is set if for at least one of the individual receive message buffers (MBCCSR*n*[MTD] = 0) the interrupt flag MBIF in the corresponding **Message Buffer Configuration, Control, Status Registers (MBCCSRn)** is equal to 1. 0 None of the individual receive message buffers has the MBIF flag asserted. 1 At least one individual receive message buffers has the MBIF flag asserted.
TBIF	**Transmit Message Buffer Interrupt Flag** — This flag is set if for at least one of the individual single or double transmit message buffers (MBCCSR*n*[MTD] = 1) the interrupt flag MBIF in the corresponding **Message Buffer Configuration, Control, Status Registers (MBCCSRn)** is equal to 1. 0 None of the individual transmit message buffers has the MBIF flag asserted. 1 At least one individual transmit message buffers has the MBIF flag asserted.

26.5.2.30 System Memory Access Time-Out Register (SYMATOR)

Base + 0x003E — Write: Disabled Mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	TIMEOUT							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0

Figure 26-30. System Memory Access Time-Out Register (SYMATOR)

Table 26-36. SYMATOR Field Descriptions

Field	Description
TIMEOUT	**System Memory Access Time-Out** — This value defines the maximum amount of time to finish a system bus access in order to ensure correct frame transmission and reception (see Section 26.6.19.2, System Bus Access Timeout).

26.5.2.31 Sync Frame Counter Register (SFCNTR)

Base + 0x0040 Additional Reset: RUN Command

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	SFEVB				SFEVA				SFODB				SFODA			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-31. Sync Frame Counter Register (SFCNTR)

This register provides the number of synchronization frames that are used for clock synchronization in the last even and in the last odd numbered communication cycle. This register is updated after the start of the NIT and before 10 MT after offset correction start.

NOTE

If the application has locked the even synchronization table at the end of the static segment of an even communication cycle, the controller will not update the fields SFEVB and SFEVA.

If the application has locked the odd synchronization table at the end of the static segment of an odd communication cycle, the controller will not update the values SFODB and SFODA.

Table 26-37. SFCNTR Field Descriptions

Field	Description
SFEVB	Sync Frames Channel B, even cycle — protocol related variable: size of (*vSyncIdListB* for even cycle) This field provides the size of the internal list of frame IDs of received synchronization frames used for clock synchronization.
SFEVB	Sync Frames Channel A, even cycle — protocol related variable: size of (*vSyncIdListA* for even cycle) This field provides the size of the internal list of frame IDs of received synchronization frames used for clock synchronization.
SFODB	Sync Frames Channel B, odd cycle — protocol related variable: size of (*vSyncIdListB* for odd cycle) This field provides the size of the internal list of frame IDs of received synchronization frames used for clock synchronization.
SFODA	Sync Frames Channel A, odd cycle — protocol related variable: size of (*vSyncIdListA* for odd cycle) This field provides the size of the internal list of frame IDs of received synchronization frames used for clock synchronization.

26.5.2.32 Sync Frame Table Offset Register (SFTOR)

Base + 0x0042 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	SFT_OFFSET[15:1]															0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-32. Sync Frame Table Offset Register (SFTOR)

This register defines the FlexRay Memory related offset for sync frame tables. For more details, see Section 26.6.12, Sync Frame ID and Sync Frame Deviation Tables.

Table 26-38. SFTOR Field Description

Field	Description
SFTOR	**Sync Frame Table Offset** — The offset of the Sync Frame Tables in the Flexray Memory. This offset is required to be 16-bit aligned. Thus STF_OFFSET[0] is always 0.

26.5.2.33 Sync Frame Table Configuration, Control, Status Register (SFTCCSR)

Base + 0x0044 Write: Normal Mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	CYCNUM						ELKS	OLKS	EVAL	OVAL	0	0	SDV EN	SID EN
W	ELKT	OLKT												OPT		
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-33. Sync Frame Table Configuration, Control, Status Register (SFTCCSR)

This register provides configuration, control, and status information related to the generation and access of the clock sync ID tables and clock sync measurement tables. For a detailed description, see Section 26.6.12, Sync Frame ID and Sync Frame Deviation Tables.

Table 26-39. SFTCCSR Field Descriptions

Field	Description
ELKT	**Even Cycle Tables Lock/Unlock Trigger** — This trigger bit is used to lock and unlock the even cycle tables. 0 No effect. 1 Triggers lock/unlock of the even cycle tables.
OLKT	**Odd Cycle Tables Lock/Unlock Trigger** — This trigger bit is used to lock and unlock the odd cycle tables. 0 No effect. 1 Triggers lock/unlock of the odd cycle tables.
CYCNUM	**Cycle Number** — This field provides the number of the cycle in which the currently locked table was recorded. If none or both tables are locked, this value is related to the even cycle table.
ELKS	**Even Cycle Tables Lock Status** — This status bit indicates whether the application has locked the even cycle tables. 0 Application has not locked the even cycle tables. 1 Application has locked the even cycle tables.
OLKS	**Odd Cycle Tables Lock Status** — This status bit indicates whether the application has locked the odd cycle tables. 0 Application has not locked the odd cycle tables. 1 Application has locked the odd cycle tables.
EVAL	**Even Cycle Tables Valid** — This status bit indicates whether the Sync Frame ID and Sync Frame Deviation Tables for the even cycle are valid. The controller clears this status bit when it starts updating the tables, and sets this bit when it has finished the table update. 0 Tables are not valid (update is ongoing). 1 Tables are valid (consistent).
OVAL	**Odd Cycle Tables Valid** — This status bit indicates whether the Sync Frame ID and Sync Frame Deviation Tables for the odd cycle are valid. The controller clears this status bit when it starts updating the tables, and sets this bit when it has finished the table update. 0 Tables are not valid (update is ongoing). 1 Tables are valid (consistent).

Table 26-39. SFTCCSR Field Descriptions (continued)

Field	Description
OPT	**One Pair Trigger** — This trigger bit controls whether the controller writes continuously or only one pair of Sync Frame Tables into the FlexRay memory. If this trigger is set to 1 while SDVEN or SIDEN is set to 1, the controller writes only one pair of the enabled Sync Frame Tables corresponding to the next even-odd-cycle pair into the FlexRay memory. In this case, the controller clears the SDVEN or SIDEN bits immediately. If this trigger is set to 0 while SDVEN or SIDEN is set to 1, the controller writes continuously the enabled Sync Frame Tables into the FlexRay memory. 0 Write continuously pairs of enabled Sync Frame Tables into FlexRay memory. 1 Write only one pair of enabled Sync Frame Tables into FlexRay memory.
SDVEN	**Sync Frame Deviation Table Enable** — This bit controls the generation of the Sync Frame Deviation Tables. The application must set this bit to request the controller to write the Sync Frame Deviation Tables into the FlexRay memory. 0 Do not write Sync Frame Deviation Tables. 1 Write Sync Frame Deviation Tables into FlexRay memory. **Note:** If SDVEN is set to 1, then SIDEN must also be set to 1.
SIDEN	**Sync Frame ID Table Enable** — This bit controls the generation of the Sync Frame ID Tables. The application must set this bit to 1 to request the controller to write the Sync Frame ID Tables into the FlexRay memory. 0 Do not write Sync Frame ID Tables. 1 Write Sync Frame ID Tables into FlexRay memory.

26.5.2.34 Sync Frame ID Rejection Filter Register (SFIDRFR)

Base + 0x0046 — 16-bit write access required — Write: Normal Mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	SYNFRID									
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-34. Sync Frame ID Rejection Filter Register (SFIDRFR)

This register defines the Sync Frame Rejection Filter ID. The application must update this register outside of the static segment. If the application updates this register in the static segment, it can appear that the controller accepts the sync frame in the current cycle.

Table 26-40. SFIDRFR Field Descriptions

Field	Description
SYNFRID	**Sync Frame Rejection ID** — This field defines the frame ID of a frame that must not be used for clock synchronization. For details see Section 26.6.15.2, Sync Frame Rejection Filtering.

26.5.2.35 Sync Frame ID Acceptance Filter Value Register (SFIDAFVR)

Base + 0x0048 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	FVAL									
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-35. Sync Frame ID Acceptance Filter Value Register (SFIDAFVR)

This register defines the sync frame acceptance filter value. For details on filtering, see Section 26.6.15, Sync Frame Filtering.

Table 26-41. SFIDAFVR Field Descriptions

Field	Description
FVAL	**Filter Value** — This field defines the value for the sync frame acceptance filtering.

26.5.2.36 Sync Frame ID Acceptance Filter Mask Register (SFIDAFMR)

Base + 0x004A Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	FMSK									
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-36. Sync Frame ID Acceptance Filter Mask Register (SFIDAFMR)

This register defines the sync frame acceptance filter mask. For details on filtering see Section 26.6.15.1, Sync Frame Acceptance Filtering.

Table 26-42. SFIDAFMR Field Descriptions

Field	Description
FMSK	**Filter Mask** — This field defines the mask for the sync frame acceptance filtering.

26.5.2.37 Network Management Vector Registers (NMVR0–NMVR5)

Base + 0x004C (NMVR0)
Base + 0x004E (NMVR1)
Base + 0x0050 (NMVR2)
Base + 0x0052 (NMVR3)
Base + 0x0054 (NMVR4)
Base + 0x0056 (NMVR5)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	NMVP[15:8]								NMVP[7:0]							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-37. Network Management Vector Registers (NMVR0–NMVR5)

Each of these six registers holds one part of the Network Management Vector. The length of the Network Management Vector is configured in the Network Management Vector Length Register (NMVLR). If NMVLR is programmed with a value that is less than 12 bytes, the remaining bytes of the Network Management Vector Registers (NMVR0–NMVR5), which are not used for the Network Management Vector accumulating, will remain 0.

The NMVR provides accrued information over all received NMVs in the last communication cycle. All NMVs received in one cycle are ORed into the NMVR. The NMVR is updated at the end of the communication cycle.

Table 26-43. NMVR[0:5] Field Descriptions

Field	Description
NMVP	**Network Management Vector Part** — The mapping between the Network Management Vector Registers (NMVR0–NMVR5) and the receive message buffer payload bytes in NMV[0:11] is depicted in **Table 26-44**.

Table 26-44. Mapping of NMVR*n* to the Received Payload Bytes NMV*n*

NMVR*n* Register	NMV*n* Received Payload
NMVR0[NMVP[15:8]]	NMV0
NMVR0[NMVP[7:0]]	NMV1
NMVR1[NMVP[15:8]]	NMV2
NMVR1[NMVP[7:0]]	NMV3
...	
NMVR5[NMVP[15:8]]	NMV10
NMVR5[NMVP[7:0]]	NMV11

26.5.2.38 Network Management Vector Length Register (NMVLR)

Base + 0x0058 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	NMVL			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-38. Network Management Vector Length Register (NMVLR)

This register defines the length of the network management vector in bytes.

Table 26-45. NMVLR Field Descriptions

Field	Description
NMVL	**Network Management Vector Length** — protocol related variable: *gNetworkManagementVectorLength* This field defines the length of the Network Management Vector in bytes. Legal values are between 0 and 12.

26.5.2.39 Timer Configuration and Control Register (TICCR)

Base + 0x005A

Write: T2_CFG: *POC:config*
T2_REP, T1_REP, T1SP, T2SP, T1TR, T2TR: Normal Mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	T2_ CFG	T2_ REP	0	0	0	T2ST	0	0	0	T1_ REP	0	0	0	T1ST
W						T2SP	T2TR							T1SP	T1TR	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-39. Timer Configuration and Control Register (TICCR)

This register is used to configure and control the two timers T1 and T2. For timer details, see Section 26.6.17, Timer Support. The Timer T1 is an absolute timer. The Timer T2 can be configured as an absolute or relative timer.

Table 26-46. TICCR Field Descriptions

Field	Description
T2_CFG	**Timer T2 Configuration** — This bit configures the timebase mode of Timer T2. 0 Timer T2 is absolute timer. 1 Timer T2 is relative timer.
T2_REP	**Timer T2 Repetitive Mode** — This bit configures the repetition mode of Timer T2. 0 Timer T2 is non repetitive. 1 Timer T2 is repetitive.
T2SP	**Timer T2 Stop** — This trigger bit is used to stop timer T2. 0 No effect. 1 Stop timer T2.
T2TR	**Timer T2 Trigger** — This trigger bit is used to start timer T2. 0 No effect. 1 Start timer T2.
T2ST	**Timer T2 State** — This status bit provides the current state of timer T2. 0 Timer T2 is idle. 1 Timer T2 is running.
T1_REP	**Timer T1 Repetitive Mode** — This bit configures the repetition mode of timer T1. 0 Timer T1 is non repetitive. 1 Timer T1 is repetitive.
T1SP	**Timer T1 Stop** — This trigger bit is used to stop timer T1. 0 No effect. 1 Stop timer T1.
T1TR	**Timer T1 Trigger** — This trigger bit is used to start timer T1. 0 No effect. 1 Start timer T1.
T1ST	**Timer T1 State** — This status bit provides the current state of timer T1. 0 Timer T1 is idle. 1 Timer T1 is running.

NOTE

Both timers are deactivated immediately when the protocol enters a state different from *POC:normal active* or *POC:normal passive*.

26.5.2.40 Timer 1 Cycle Set Register (TI1CYSR)

Base + 0x005C — Write: Anytime

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	T1_CYC_VAL						0	0	T1_CYC_MSK					
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-40. Timer 1 Cycle Set Register (TI1CYSR)

This register defines the cycle filter value and the cycle filter mask for timer T1. For a detailed description of timer T1, refer to Section 26.6.17.1, Absolute Timer T1.

Table 26-47. TI1CYSR Field Descriptions

Field	Description
T1_CYC_VAL	**Timer T1 Cycle Filter Value** — This field defines the cycle filter value for timer T1.
T1_CYC_MSK	**Timer T1 Cycle Filter Mask** — This field defines the cycle filter mask for timer T1.

NOTE

If the application modifies the value in this register while the timer is running, the change becomes effective immediately and timer T1 will expire according to the changed value.

26.5.2.41 Timer 1 Macrotick Offset Register (TI1MTOR)

Base + 0x005E — Write: Anytime

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	T1_MTOFFSET													
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-41. Timer 1 Macrotick Offset Register (TI1MTOR)

This register holds the macrotick offset value for timer T1. For a detailed description of timer T1, refer to Section 26.6.17.1, Absolute Timer T1.

Table 26-48. TI1MTOR Field Descriptions

Field	Description
T1_MTOFFSET	**Timer 1 Macrotick Offset** — This field defines the macrotick offset value for timer 1.

NOTE

If the application modifies the value in this register while the timer is running, the change becomes effective immediately and timer T1 will expire according to the changed value.

26.5.2.42 Timer 2 Configuration Register 0 (TI2CR0)

Base + 0x0060 — Write: Anytime

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	R*				T2_CYC_VAL				R*				T2_CYC_MSK			
R W	T2_MTCNT[31:16]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-42. Timer 2 Configuration Register 0 (TI2CR0)

The content of this register depends on the value of the T2_CFG bit in the Timer Configuration and Control Register (TICCR). For a detailed description of timer T2, refer to Section 26.6.17.2, Absolute / Relative Timer T2.

Table 26-49. TI2CR0 Field Descriptions

Field	Description
Fields for absolute timer T2 (TICCR[T2_CFG] = 0)	
T2_CYC_VAL	**Timer T2 Cycle Filter Value** — This field defines the cycle filter value for timer T2.
T2_CYC_MSK	**Timer T2 Cycle Filter Mask** — This field defines the cycle filter mask for timer T2.
Fields for relative timer T2 (TICCR[T2_CFG = 1)	
T2_MTCNT[31:16]	**Timer T2 Macrotick High Word** — This field defines the high word of the macrotick count for timer T2.

NOTE

If timer T2 is configured as an *absolute* timer and the application modifies the values in this register while the timer is running, the change becomes effective immediately and timer T2 will expire according to the changed values.

If timer T2 is configured as a *relative* timer and the application changes the values in this register while the timer is running, the change becomes effective when the timer has expired according to the old values.

26.5.2.43 Timer 2 Configuration Register 1 (TI2CR1)

Base + 0x0062 — Write: Anytime

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	R*				T2_MTOFFSET											
R W	T2_MTCNT[15:0]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-43. Timer 2 Configuration Register 1 (TI2CR1)

The content of this register depends on the value of the T2_CFG bit in the Timer Configuration and Control Register (TICCR). For a detailed description of timer T2, refer to Section 26.6.17.2, Absolute / Relative Timer T2.

Table 26-50. TI2CR1 Field Descriptions

Field	Description
Fields for absolute timer T2 (TICCR[T2_CFG] = 0)	
T2_MTOFFSET	**Timer T2 Macrotick Offset** — This field holds the macrotick offset value for timer T2.
Fields for relative timer T2 (TICCR[T2_CFG] = 1)	
T2_MTCNT[15:0]	**Timer T2 Macrotick Low Word** — This field defines the low word of the macrotick value for timer T2.

NOTE

If timer T2 is configured as an *absolute* timer and the application modifies the values in this register while the timer is running, the change becomes effective immediately and the timer T2 will expire according to the changed values.

If timer T2 is configured as a *relative* timer and the application changes the values in this register while the timer is running, the change becomes effective when the timer has expired according to the old values.

26.5.2.44 Slot Status Selection Register (SSSR)

Base + 0x0064 — 16-bit write access required — Write: Anytime

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	SEL		0	SLOTNUMBER										
W	WMD															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-44. Slot Status Selection Register (SSSR)

This register is used to access the four internal non memory-mapped slot status selection registers SSSR0 to SSSR3. Each internal registers selects a slot, or symbol window/NIT, whose status vector will be saved in the corresponding Slot Status Registers (SSR0–SSR7) according to Table 26-52. For a detailed description of slot status monitoring, refer to Section 26.6.18, Slot Status Monitoring.

Table 26-51. SSSR Field Descriptions

Field	Description
WMD	**Write Mode** — This control bit defines the write mode of this register. 0 Write to all fields in this register on write access. 1 Write to SEL field only on write access.

Table 26-51. SSSR Field Descriptions (continued)

Field	Description
SEL	**Selector** — This field selects one of the four internal slot status selection registers for access. 00 Select SSSR0. 01 Select SSSR1. 10 Select SSSR2. 11 Select SSSR3.
SLOTNUMBER	**Slot Number** — This field specifies the number of the slot whose status will be saved in the corresponding slot status registers. **Note:** If this value is set to 0, the related slot status register provides the status of the symbol window after the NIT start, and provides the status of the NIT after the cycle start.

Table 26-52. Mapping Between SSSR*n* and SSR*n*

Internal Slot Status Selection Register	Write the Slot Status of the Slot Selected by SSSR*n* for each			
	Even Communication Cycle		Odd Communication Cycle	
	For Channel B to	For Channel A to	For Channel B to	For Channel A to
SSSR0	SSR0[15:8]	SSR0[7:0]	SSR1[15:8]	SSR1[7:0]
SSSR1	SSR2[15:8]	SSR2[7:0]	SSR3[15:8]	SSR3[7:0]
SSSR2	SSR4[15:8]	SSR4[7:0]	SSR5[15:8]	SSR5[7:0]
SSSR3	SSR6[15:8]	SSR6[7:0]	SSR7[15:8]	SSR7[7:0]

26.5.2.45 Slot Status Counter Condition Register (SSCCR)

Base + 0x0066 — 16-bit write access required — Write: Anytime

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	SEL		0	CNTCFG		MCY	VFR	SYF	NUF	SUF	STATUSMASK[3:0]			
W	WMD															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-45. Slot Status Counter Condition Register (SSCCR)

This register is used to access and program the four internal non-memory mapped Slot Status Counter Condition Registers SSCCR0 to SSCCR3. Each of these four internal slot status counter condition registers defines the mode and the conditions for incrementing the counter in the corresponding Slot Status Counter Registers (SSCR0–SSCR3). The correspondence is given in Table 26-54. For a detailed description of slot status counters, refer to Section 26.6.18.4, Slot Status Counter Registers.

Table 26-53. SSCCR Field Descriptions

Field	Description
WMD	**Write Mode** — This control bit defines the write mode of this register. 0 Write to all fields in this register on write access. 1 Write to SEL field only on write access.
SEL	**Selector** — This field selects one of the four internal slot counter condition registers for access. 00 Select SSCCR0. 01 Select SSCCR1. 10 Select SSCCR2. 11 Select SSCCR3.
CNTCFG	**Counter Configuration** — These bit field controls the channel related incrementing of the slot status counter. 00 Increment by 1 if condition is fulfilled on channel A. 01 Increment by 1 if condition is fulfilled on channel B. 10 Increment by 1 if condition is fulfilled on at least one channel. 11 Increment by 2 if condition is fulfilled on both channels. Increment by 1 if condition is fulfilled on only one channel.
MCY	**Multi Cycle Selection** — This bit defines whether the slot status counter accumulates over multiple communication cycles or provides information for the previous communication cycle only. 0 The Slot Status Counter provides information for the previous communication cycle only. 1 The Slot Status Counter accumulates over multiple communication cycles.
VFR	**Valid Frame Restriction** — This bit is used to restrict the counter to received valid frames. 0 The counter is not restricted to valid frames only. 1 The counter is restricted to valid frames only.
SYF	**Sync Frame Restriction** — This bit is used to restrict the counter to received frames with the sync frame indicator bit set to 1. 0 The counter is not restricted with respect to the sync frame indicator bit. 1 The counter is restricted to frames with the sync frame indicator bit set to 1.
NUF	**Null Frame Restriction** — This bit is used to restrict the counter to received frames with the null frame indicator bit set to 0. 0 The counter is not restricted with respect to the null frame indicator bit. 1 The counter is restricted to frames with the null frame indicator bit set to 0.
SUF	**Startup Frame Restriction** — This bit is used to restrict the counter to received frames with the startup frame indicator bit set to 1. 0 The counter is not restricted with respect to the startup frame indicator bit. 1 The counter is restricted to received frames with the startup frame indicator bit set to 1.
STATUS MASK[3:0]	**Slot Status Mask** — This bit field is used to enable the counter with respect to the four slot status error indicator bits. **STATUSMASK[3]** – This bit enables the counting for slots with the syntax error indicator bit set to 1. **STATUSMASK[2]** – This bit enables the counting for slots with the content error indicator bit set to 1. **STATUSMASK[1]** – This bit enables the counting for slots with the boundary violation indicator bit set to 1. **STATUSMASK[0]** – This bit enables the counting for slots with the transmission conflict indicator bit set to 1.

Table 26-54. Mapping between internal SSCCR*n* and SSCR*n*

Condition Register	Condition Defined for Register
SSCCR0	SSCR0
SSCCR1	SSCR1
SSCCR2	SSCR2
SSCCR3	SSCR3

26.5.2.46 Slot Status Registers (SSR0–SSR7)

Base + 0x0068 (SSR0)
Base + 0x006A (SSR1)
Base + 0x006C (SSR2)
Base + 0x006E (SSR3)
Base + 0x0070 (SSR4)
Base + 0x0072 (SSR5)
Base + 0x0074 (SSR6)
Base + 0x0076 (SSR7)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	VFB	SYB	NFB	SUB	SEB	CEB	BVB	TCB	VFA	SYA	NFA	SUA	SEA	CEA	BVA	TCA
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-46. Slot Status Registers (SSR0–SSR7)

Each of these eight registers holds the status vector of the slot specified in the corresponding internal slot status selection register, which can be programmed using the Slot Status Selection Register (SSSR). Each register is updated after the end of the corresponding slot as shown in Figure 26-148. The register bits are directly related to the protocol variables and described in more detail in Section 26.6.18, Slot Status Monitoring.

Table 26-55. SSR0–SSR7 Field Descriptions

Field	Description
VFB	**Valid Frame on Channel B** — protocol related variable: *vSS!ValidFrame* channel B 0 *vSS!ValidFrame* = 0. 1 *vSS!ValidFrame* = 1.
SYB	**Sync Frame Indicator Channel B** — protocol related variable: *vRF!Header!SyFIndicator* channel B 0 *vRF!Header!SyFIndicator* = 0. 1 *vRF!Header!SyFIndicator* = 1.
NFB	**Null Frame Indicator Channel B** — protocol related variable: *vRF!Header!NFIndicator* channel B 0 *vRF!Header!NFIndicator* = 0. 1 *vRF!Header!NFIndicator* = 1.
SUB	**Startup Frame Indicator Channel B** — protocol related variable: *vRF!Header!SuFIndicator* channel B 0 *vRF!Header!SuFIndicator* = 0. 1 *vRF!Header!SuFIndicator* = 1.
SEB	**Syntax Error on Channel B** — protocol related variable: *vSS!SyntaxError* channel B 0 *vSS!SyntaxError* = 0. 1 *vSS!SyntaxError* = 1.
CEB	**Content Error on Channel B** — protocol related variable: *vSS!ContentError* channel B 0 *vSS!ContentError* = 0. 1 *vSS!ContentError* = 1.
BVB	**Boundary Violation on Channel B** — protocol related variable: *vSS!BViolation* channel B 0 *vSS!BViolation* = 0. 1 *vSS!BViolation* = 1.
TCB	**Transmission Conflict on Channel B** — protocol related variable: *vSS!TxConflict* channel B 0 *vSS!TxConflict* = 0. 1 *vSS!TxConflict* = 1.

Table 26-55. SSR0–SSR7 Field Descriptions (continued)

Field	Description
VFA	**Valid Frame on Channel A** — protocol related variable: *vSS!ValidFrame* channel A 0 *vSS!ValidFrame* = 0. 1 *vSS!ValidFrame* = 1.
SYA	**Sync Frame Indicator Channel A** — protocol related variable: *vRF!Header!SyFIndicator* channel A 0 *vRF!Header!SyFIndicator* = 0. 1 *vRF!Header!SyFIndicator* = 1.
NFA	**Null Frame Indicator Channel A** — protocol related variable: *vRF!Header!NFIndicator* channel A 0 *vRF!Header!NFIndicator* = 0. 1 *vRF!Header!NFIndicator* = 1.
SUA	**Startup Frame Indicator Channel A** — protocol related variable: *vRF!Header!SuFIndicator* channel A 0 *vRF!Header!SuFIndicator* = 0. 1 *vRF!Header!SuFIndicator* = 1.
SEA	**Syntax Error on Channel A** — protocol related variable: *vSS!SyntaxError* channel A 0 *vSS!SyntaxError* = 0. 1 *vSS!SyntaxError* = 1.
CEA	**Content Error on Channel A** — protocol related variable: *vSS!ContentError* channel A 0 *vSS!ContentError* = 0. 1 *vSS!ContentError* = 1.
BVA	**Boundary Violation on Channel A** — protocol related variable: *vSS!BViolation* channel A 0 *vSS!BViolation* = 0. 1 *vSS!BViolation* = 1.
TCA	**Transmission Conflict on Channel A** — protocol related variable: *vSS!TxConflict* channel A 0 *vSS!TxConflict* = 0. 1 *vSS!TxConflict* = 1.

26.5.2.47 Slot Status Counter Registers (SSCR0–SSCR3)

Base + 0x0078 (SSCR0)
Base + 0x007A (SSCR1)
Base + 0x007C (SSCR2)
Base + 0x007E (SSCR3)

Additional Reset: RUN Command

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	SLOTSTATUSCNT															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-47. Slot Status Counter Registers (SSCR0–SSCR3)

Each of these four registers provides the slot status counter value for the previous communication cycle(s) and is updated at the cycle start. The provided value depends on the control bits and fields in the related internal slot status counter condition register SSCCR*n*, which can be programmed by using the Slot Status Counter Condition Register (SSCCR). For more details, see Section 26.6.18.4, Slot Status Counter Registers.

NOTE

If the counter has reached its maximum value 0xFFFF and is in the multicycle mode (SSCCR*n*[MCY] = 1), the counter is not reset to 0x0000. The application can reset the counter by clearing the SSCCR*n*[MCY] bit and waiting for the next cycle start, when the controller clears the counter. Subsequently, the counter can be set into the multicycle mode again.

Table 26-56. SSCR0–SSCR3 Field Descriptions

Field	Description
SLOTSTATUSCNT	**Slot Status Counter** — This field provides the current value of the Slot Status Counter.

26.5.2.48 MTS A Configuration Register (MTSACFR)

Base + 0x0080

Write: MTE: Anytime
CYCCNTMSK,CYCCNTVAL:*POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	MTE	0	CYCCNTMSK						0	0	CYCCNTVAL					
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-48. MTS A Configuration Register (MTSACFR)

This register controls the transmission of the Media Access Test Symbol MTS on channel A. For more details, see Section 26.6.13, MTS Generation.

Table 26-57. MTSACFR Field Descriptions

Field	Description
MTE	**Media Access Test Symbol Transmission Enable** — This control bit is used to enable and disable the transmission of the Media Access Test Symbol in the selected set of cycles. 0 MTS transmission disabled. 1 MTS transmission enabled.
CYCCNTMSK	Cycle Counter Mask — This field provides the filter mask for the MTS cycle count filter.
CYCCNTVAL	**Cycle Counter Value** — This field provides the filter value for the MTS cycle count filter.

26.5.2.49 MTS B Configuration Register (MTSBCFR)

Base + 0x0082

Write: MTE: Anytime
CYCCNTMSK,CYCCNTVAL:*POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	MTE	0	CYCCNTMSK						0	0	CYCCNTVAL					
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-49. MTS B Configuration Register (MTSBCFR)

This register controls the transmission of the Media Access Test Symbol MTS on channel B. For more details, see Section 26.6.13, MTS Generation.

Table 26-58. MTSBCFR Field Descriptions

Field	Description
MTE	**Media Access Test Symbol Transmission Enable** — This control bit is used to enable and disable the transmission of the Media Access Test Symbol in the selected set of cycles. 0 MTS transmission disabled. 1 MTS transmission enabled.
CYCCNTMSK	Cycle Counter Mask — This field provides the filter mask for the MTS cycle count filter.
CYCCNTVAL	**Cycle Counter Value** — This field provides the filter value for the MTS cycle count filter.

26.5.2.50 Receive Shadow Buffer Index Register (RSBIR)

Base + 0x0084 — 16-bit write access required — Write: WMD, SEL: Any Time; RSBIDX: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	SEL		0	0	0	0	RSBIDX							
W	WMD															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-50. Receive Shadow Buffer Index Register (RSBIR)

This register is used to provide and retrieve the indices of the message buffer header fields currently associated with the receive shadow buffers. For more details on the receive shadow buffer concept, refer to Section 26.6.6.3.5, Receive Shadow Buffers Concept.

Table 26-59. RSBIR Field Descriptions

Field	Description
WMD	**Write Mode** — This bit controls the write mode for this register. 0 update SEL and RSBIDX field on register write 1 update only SEL field on register write
SEL	**Selector** — This field is used to select the internal receive shadow buffer index register for access. 00 RSBIR_A1 — Receive shadow buffer index register for channel A, segment 1. 01 RSBIR_A2 — Receive shadow buffer index register for channel A, segment 2. 10 RSBIR_B1 — Receive shadow buffer index register for channel B, segment 1. 11 RSBIR_B2 — Receive shadow buffer index register for channel B, segment 2.
RSBIDX	**Receive Shadow Buffer Index** — This field contains the current index of the message buffer header field of the receive shadow message buffer selected by the SEL field. The controller uses this index to determine the physical location of the shadow buffer header field in the FlexRay memory. The controller will update this field during receive operation.The application provides initial message buffer header index value in the configuration phase. controller: Updates the message buffer header index after successful reception. Application: Provides initial message buffer header index.

26.5.2.51 Receive FIFO System Memory Base Address Register (RFSYMBADR)

Base + 0x00E8 Write: Disabled Mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	SMBA[31:16]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-51. Receive FIFO System Memory Base Address High Register (RFSYMBADHR)

Base + 0x00EA Write: Disabled Mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	SMBA[15:4]												0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-52. Receive FIFO System Memory Base Address Low Register (RFSYMBADLR)

These registers define the system memory base address for the receive FIFO if the FIFO address mode bit MCR[FAM] is set to 1. The system memory base address is used by the BMIF to calculate the physical memory address for system memory accesses for the FIFOs.

Table 26-60. RFSYMBADR Field Descriptions

Field	Description
SMBA	**System Memory Base Address** — This is the value of the system memory base address for the receive FIFO if the FIFO address mode bit MCR[FAM] is set to 1. It is defines as a byte address.

26.5.2.52 Receive FIFO Periodic Timer Register (RFPTR)

Base + 0x00EC Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	PTD													
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-53. Receive FIFO Periodic Timer Register (RFPTR)

This register holds periodic timer duration for the periodic FIFO timer. The periodic timer applies to both FIFOs (see Section 26.6.9.3, FIFO Periodic Timer).

Table 26-61. RFPTIR Field Descriptions

Field	Description
PTD	**Periodic Timer Duration** — This value defines the periodic timer duration in terms of macroticks. 0000 Timer stays expired. 3FFF Timer never expires. other Timer expires after specified number of macroticks, expires and is restarted at each cycle start.

26.5.2.53 Receive FIFO Watermark and Selection Register (RFWMSR)

Base + 0x0086 — Write: *WM_A/WM_B: POC:config, SEL: Anytime*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	WM_A/WM_B								0	0	0	0	0	0	0	SEL
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-54. Receive FIFO Watermark and Selection Register (RFWMSR)

This register is used to

- select a receiver FIFO for subsequent programming access through the receiver FIFO configuration registers summarized in Table 26-62.
- to define the watermark for the selected FIFO.

Table 26-62. SEL Controlled Receiver FIFO Registers

Register
Receive FIFO Start Index Register (RFSIR)
Receive FIFO Depth and Size Register (RFDSR)
Receive FIFO Message ID Acceptance Filter Value Register (RFMIDAFVR)
Receive FIFO Message ID Acceptance Filter Mask Register (RFMIAFMR)
Receive FIFO Frame ID Rejection Filter Value Register (RFFIDRFVR)
Receive FIFO Frame ID Rejection Filter Mask Register (RFFIDRFMR)
Receive FIFO Range Filter Configuration Register (RFRFCFR)
Receive FIFO Range Filter Control Register (RFRFCTR)

Table 26-63. RFSR Field Descriptions

Field	Description
WM_A WM_B	**Watermark** — This field defines the watermark value for the selected FIFO. This value is used to control the generation of the almost full interrupt flags.
SEL	**Select** — This control bit selects the receiver FIFO for subsequent programming. 0 Receiver FIFO for channel A selected. 1 Receiver FIFO for channel B selected.

26.5.2.54 Receive FIFO Start Index Register (RFSIR)

Base + 0x0088 — Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	$SIDX_A$/$SIDX_B$									
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-55. Receive FIFO Start Index Register (RFSIR)

This register defines the message buffer header index of the first message buffer of the selected FIFO.

Table 26-64. RFSIR Field Descriptions

Field	Description
$SIDX_A$ $SIDX_B$	**Start Index** — This field defines the number of the message buffer header field of the first message buffer of the selected FIFO. The controller uses the value of the SIDX field to determine the physical location of the receiver FIFO's first message buffer header field.

26.5.2.55 Receive FIFO Depth and Size Register (RFDSR)

Base + 0x008A — Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	FIFO_DEPTH$_A$/FIFO_DEPTH$_B$								0	ENTRY_SIZE$_A$/ENTRY_SIZE$_B$						
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-56. Receive FIFO Depth and Size Register (RFDSR)

This register defines the structure of the selected FIFO, i.e.,the number of entries and the size of each entry.

Table 26-65. RFDSR Field Descriptions

Field	Description
$FIFO\_DEPTH_A$ $FIFO\_DEPTH_B$	**FIFO Depth** — This field defines the depth of the selected FIFO, i.e.,the number of entries.
$ENTRY\_SIZE_A$ $ENTRY\_SIZE_B$	**Entry Size** — This field defines the size of the frame data sections for the selected FIFO in 2 byte entities.

26.5.2.56 Receive FIFO A Read Index Register (RFARIR)

Base + 0x008C

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	RDIDX									
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-57. Receive FIFO A Read Index Register (RFARIR)

This register provides the message buffer header index of the next available FIFO A entry that the application can read.

Table 26-66. RFARIR Field Descriptions

Field	Description
RDIDX	**Read Index** — This field provides the message buffer header index of the next available FIFO message buffer that the application can read. If the old style FIFO mode is configured (MCR.FIMD = 0), the controller updates this index by 1 entry, when the application writes to the FAFAIF flag in the Global Interrupt Flag and Enable Register (GIFER). If the new style FIFO mode is configured (MCR.FIMD = 1), the controller updates this index by PCA entries, when the application writes to the Receive FIFO Fill Level and POP Count Register (RFFLPCR).

NOTE

If the FIFO is empty, the RDIDX field points to an physical message buffer with invalid content.

26.5.2.57 Receive FIFO B Read Index Register (RFBRIR)

Base + 0x008E

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	RDIDX									
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-58. Receive FIFO B Read Index Register (RFBRIR)

This register provides the message buffer header index of the next available FIFO B entry that the application can read.

Table 26-67. RFBRIR Field Descriptions

Field	Description
RDIDX	**Read Index** — This field provides the message buffer header index of the next available FIFO message buffer that the application can read.

NOTE

If the FIFO is empty, the RDIDX field points to an physical message buffer with invalid content.

26.5.2.58 Receive FIFO Fill Level and POP Count Register (RFFLPCR)

Base + 0x00EE

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	FLB								FLA							
W	PCB								PCA							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-59. Receive FIFO Fill Level and POP Count Register (RFFLPCR)

This register provides the current fill level of the two receiver FIFOs and is used to pop a number of entries from the FIFOs.

Table 26-68. RFFLPCR Field Descriptions

Field	Description
FLB	**Fill Level FIFO B** — This field provides the current number of entries in the FIFO B.
FLA	**Fill Level FIFO A**— This field provides the current number of entries in the FIFO A.
PCB	**Pop Count FIFO B** — This field defines the number of entries to be removed from FIFO B.
PCA	**Pop Count FIFO A**— This field defines the number of entries to be removed from FIFO A.

NOTE

If the pop count value PCA/PCB is greater than the current FIFO fill level FLB/FLA, than the FIFO is empty after the update. No notification is given that not the required number of entries was removed.

26.5.2.59 Receive FIFO Message ID Acceptance Filter Value Register (RFMIDAFVR)

Base + 0x0090 — Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	$MIDAFVAL_A/MIDAFVAL_B$															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-60. Receive FIFO Message ID Acceptance Filter Value Register (RFMIDAFVR)

This register defines the filter value for the message ID acceptance filter of the selected FIFO. For details on message ID filtering see Section 26.6.9.9, FIFO Filtering.

Table 26-69. RFMIDAFVR Field Descriptions

Field	Description
$MIDAFVAL_A$ $MIDAFVAL_B$	**Message ID Acceptance Filter Value** — Filter value for the message ID acceptance filter.

26.5.2.60 Receive FIFO Message ID Acceptance Filter Mask Register (RFMIAFMR)

Base + 0x0092 — Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	$MIDAFMSK_A/MIDAFMSK_B$															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-61. Receive FIFO Message ID Acceptance Filter Mask Register (RFMIAFMR)

This register defines the filter mask for the message ID acceptance filter of the selected FIFO. For details on message ID filtering see Section 26.6.9.9, FIFO Filtering.

Table 26-70. RFMIAFMR Field Descriptions

Field	Description
$MIDAFMSK_A$ $MIDAFMSK_B$	**Message ID Acceptance Filter Mask** — Filter mask for the message ID acceptance filter.

26.5.2.61 Receive FIFO Frame ID Rejection Filter Value Register (RFFIDRFVR)

Base + 0x0094 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	FIDRFVAL$_A$/FIDRFVAL$_B$										
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-62. Receive FIFO Frame ID Rejection Filter Value Register (RFFIDRFVR)

This register defines the filter value for the frame ID rejection filter of the selected FIFO. For details on frame ID filtering see Section 26.6.9.9, FIFO Filtering.

Table 26-71. RFFIDRFVR Field Descriptions

Field	Description
FIDRFVAL$_A$ FIDRFVAL$_B$	**Frame ID Rejection Filter Value** — Filter value for the frame ID rejection filter.

26.5.2.62 Receive FIFO Frame ID Rejection Filter Mask Register (RFFIDRFMR)

Base + 0x0096 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	FIDRFMSK$_A$/FIDRFMSK$_B$										
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-63. Receive FIFO Frame ID Rejection Filter Mask Register (RFFIDRFMR)

This register defines the filter mask for the frame ID rejection filter of the selected FIFO. For details on frame ID filtering see Section 26.6.9.9, FIFO Filtering.

Table 26-72. RFFIDRFMR Field Descriptions

Field	Description
FIDRFMSK	**Frame ID Rejection Filter Mask** — Filter mask for the frame ID rejection filter.

26.5.2.63 Receive FIFO Range Filter Configuration Register (RFRFCFR)

Base + 0x0098 16-bit write access required Write: WMD, IBD, SEL: Any Time
SID: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	IBD	SEL		0	SID$_A$/SID$_B$										
W	WMD															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-64. Receive FIFO Range Filter Configuration Register (RFRFCFR)

This register provides access to the four internal frame ID range filter boundary registers of the selected FIFO. For details on frame ID range filter see Section 26.6.9.9, FIFO Filtering.

Table 26-73. RFRFCFR Field Descriptions

Field	Description
WMD	**Write Mode** — This control bit defines the write mode of this register. 0 Write to all fields in this register on write access. 1 Write to SEL and IBD field only on write access.
IBD	**Interval Boundary** — This control bit selects the interval boundary to be programmed with the SID value. 0 program lower interval boundary. 1 program upper interval boundary.
SEL	**Filter Selector** — This control field selects the frame ID range filter to be accessed. 00 Select frame ID range filter 0. 01 Select frame ID range filter 1. 10 Select frame ID range filter 2. 11 Select frame ID range filter 3.
SID_A SID_B	**Slot ID —** Defines the IBD-selected frame ID boundary value for the SEL-selected range filter.

26.5.2.64 Receive FIFO Range Filter Control Register (RFRFCTR)

Base + 0x009A — Write: Anytime

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	F3MD	F2MD	F1MD	F0MD	0	0	0	0	F3EN	F2EN	F1EN	F0EN
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-65. Receive FIFO Range Filter Control Register (RFRFCTR)

This register is used to enable and disable each frame ID range filter and to define whether it is running as acceptance or rejection filter.

Table 26-74. RFRFCTR Field Descriptions

Field	Description
F3MD	**Range Filter 3 Mode** — This control bit defines the filter mode of the frame ID range filter 3. 0 Range filter 3 runs as acceptance filter. 1 Range filter 3 runs as rejection filter.
F2MD	**Range Filter 2 Mode** — This control bit defines the filter mode of the frame ID range filter 2. 0 Range filter 2 runs as acceptance filter. 1 Range filter 2 runs as rejection filter.
F1MD	**Range Filter 1 Mode** — This control bit defines the filter mode of the frame ID range filter 1. 0 Range filter 1 runs as acceptance filter. 1 Range filter 1 runs as rejection filter.
F0MD	**Range Filter 0 Mode** — This control bit defines the filter mode of the frame ID range filter 0. 0 Range filter 0 runs as acceptance filter. 1 Range filter 0 runs as rejection filter.
F3EN	**Range Filter 3 Enable** — This control bit is used to enable and disable the frame ID range filter 3. 0 Range filter 3 disabled. 1 Range filter 3 enabled.
F2EN	**Range Filter 2 Enable** — This control bit is used to enable and disable the frame ID range filter 2. 0 Range filter 2 disabled. 1 Range filter 2 enabled.

Table 26-74. RFRFCTR Field Descriptions (continued)

Field	Description
F1EN	**Range Filter 1 Enable** — This control bit is used to enable and disable the frame ID range filter 1. 0 Range filter 1 disabled. 1 Range filter 1 enabled.
F0EN	**Range Filter 0 Enable** — This control bit is used to enable and disable the frame ID range filter 0. 0 Range filter 0 disabled. 1 Range filter 0 enabled.

26.5.2.65 Last Dynamic Transmit Slot Channel A Register (LDTXSLAR)

Base + 0x009C

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	LASTDYNTXSLOTA										
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-66. Last Dynamic Slot Channel A Register (LDTXSLAR)

This register provides the number of the last transmission slot in the dynamic segment for channel A. This register is updated after the end of the dynamic segment and before the start of the next communication cycle.

Table 26-75. LDTXSLAR Field Descriptions

Field	Description
LASTDYNTXSLOTA	**Last Dynamic Transmission Slot Channel A** — protocol related variable: *zLastDynTxSlot* channel A Number of the last transmission slot in the dynamic segment for channel A. If no frame was transmitted during the dynamic segment on channel A, the value of this field is set to 0.

26.5.2.66 Last Dynamic Transmit Slot Channel B Register (LDTXSLBR)

Base + 0x009E

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	LASTDYNTXSLOTB										
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-67. Last Dynamic Slot Channel B Register (LDTXSLBR)

This register provides the number of the last transmission slot in the dynamic segment for channel B. This register is updated after the end of the dynamic segment and before the start of the next communication cycle.

Table 26-76. LDTXSLBR Field Descriptions

Field	Description
LASTDYNTXSLOTB	**Last Dynamic Transmission Slot Channel B** — protocol related variable: *zLastDynTxSlot* channel B Number of the last transmission slot in the dynamic segment for channel B. If no frame was transmitted during the dynamic segment on channel B the value of this field is set to 0.

26.5.2.67 Protocol Configuration Registers

The following configuration registers provide the necessary configuration information to the protocol engine. The individual values in the registers are described in Table 26-77. For more details about the FlexRay related configuration parameters and the allowed parameter ranges, see *FlexRay Communications System Protocol Specification, Version 2.1 Rev A.*

Table 26-77. Protocol Configuration Register Fields

Name	Description[1]	Min	Max	Unit	PCR
coldstart_attempts	gColdstartAttempts			number	3
action_point_offset	*gdActionPointOffset* - 1			MT	0
cas_rx_low_max	*gdCASRxLowMax* - 1			*gdBit*	4
dynamic_slot_idle_phase	gdDynamicSlotIdlePhase			minislot	28
minislot_action_point_offset	*gdMinislotActionPointOffset* - 1			MT	3
minislot_after_action_point	*gdMinislot* - *gdMinislotActionPointOffset* - 1			MT	2
static_slot_length	gdStaticSlot			MT	0
static_slot_after_action_point	*gdStaticSlot* - *gdActionPointOffset* - 1			MT	13
symbol_window_exists	*gdSymbolWindow*!=0	0	1	bool	9
symbol_window_after_action_point	*gdSymbolWindow* - *gdActionPointOffset* - 1			MT	6
tss_transmitter	gdTSSTransmitter			*gdBit*	5
wakeup_symbol_rx_idle	gdWakeupSymbolRxIdle			*gdBit*	5
wakeup_symbol_rx_low	gdWakeupSymbolRxLow			*gdBit*	3
wakeup_symbol_rx_window	gdWakeupSymbolRxWindow			*gdBit*	4
wakeup_symbol_tx_idle	gdWakeupSymbolTxIdle			*gdBit*	8
wakeup_symbol_tx_low	gdWakeupSymbolTxLow			*gdBit*	5
noise_listen_timeout	(*gListenNoise* * *pdListenTimeout*) - 1			µT	16/17
macro_initial_offset_a	pMacroInitialOffset[A]			MT	6
macro_initial_offset_b	pMacroInitialOffset[B]			MT	16
macro_per_cycle	gMacroPerCycle			MT	10
macro_after_first_static_slot	*gMacroPerCycle* - *gdStaticSlot*			MT	1
macro_after_offset_correction	*gMacroPerCycle* - *gOffsetCorrectionStart*			MT	28
max_without_clock_correction_fatal	gMaxWithoutClockCorrectionFatal			cyclepairs	8
max_without_clock_correction_passive	gMaxWithoutClockCorrectionPassive			cyclepairs	8
minislot_exists	*gNumberOfMinislots*!=0	0	1	bool	9
minislots_max	*gNumberOfMinislots* - 1			minislot	29
number_of_static_slots	gNumberOfStaticSlots			static slot	2
offset_correction_start	gOffsetCorrectionStart			MT	11
payload_length_static	gPayloadLengthStatic			2-bytes	19
max_payload_length_dynamic	pPayloadLengthDynMax			2-bytes	24
first_minislot_action_point_offset	max(*gdActionPointOffset*, *gdMinislotActionPointOffset*) - 1			MT	13
allow_halt_due_to_clock	pAllowHaltDueToClock			bool	26
allow_passive_to_active	pAllowPassiveToActive			cyclepairs	12

Table 26-77. Protocol Configuration Register Fields (continued)

Name	Description[1]	Min	Max	Unit	PCR
cluster_drift_damping	pClusterDriftDamping			µT	24
comp_accepted_startup_range_a	*pdAcceptedStartupRange* - *pDelayCompensation[A]*			µT	22
comp_accepted_startup_range_b	*pdAcceptedStartupRange* - *pDelayCompensation[B]*			µT	26
listen_timeout	*pdListenTimeout* - 1			µT	14/15
key_slot_id	pKeySlotId			number	18
key_slot_used_for_startup	pKeySlotUsedForStartup			bool	11
key_slot_used_for_sync	pKeySlotUsedForSync			bool	11
latest_tx	*gNumberOfMinislots* - *pLatestTx*			minislot	21
sync_node_max	gSyncNodeMax			number	30
micro_initial_offset_a	pMicroInitialOffset[A]			µT	20
micro_initial_offset_b	pMicroInitialOffset[B]			µT	20
micro_per_cycle	pMicroPerCycle			µT	22/23
micro_per_cycle_min	pMicroPerCycle - pdMaxDrift			µT	24/25
micro_per_cycle_max	pMicroPerCycle + pdMaxDrift			µT	26/27
micro_per_macro_nom_half	round(*pMicroPerMacroNom* / 2)			µT	7
offset_correction_out	pOffsetCorrectionOut			µT	9
rate_correction_out	pRateCorrectionOut			µT	14
single_slot_enabled	pSingleSlotEnabled			bool	10
wakeup_channel	pWakeupChannel	see Table 26-78			10
wakeup_pattern	pWakeupPattern			number	18
decoding_correction_a	*pDecodingCorrection* + *pDelayCompensation[A]* + 2			µT	19
decoding_correction_b	*pDecodingCorrection* + *pDelayCompensation[B]* + 2			µT	7
key_slot_header_crc	header CRC for key slot	0x000	0x7FF	number	12
extern_offset_correction	pExternOffsetCorrection			µT	29
extern_rate_correction	pExternRateCorrection			µT	21

[1] See *FlexRay Communications System Protocol Specification, Version 2.1 Rev A* for detailed protocol parameter definitions

Table 26-78. Wakeup Channel Selection

wakeup_channel	Wakeup Channel
0	A
1	B

26.5.2.67.1 Protocol Configuration Register 0 (PCR0)

Base + 0x00A0 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	action_point_offset								static_slot_length							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-68. Protocol Configuration Register 0 (PCR0)

26.5.2.67.2 Protocol Configuration Register 1 (PCR1)

Base + 0x00A2 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	macro_after_first_static_slot													
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-69. Protocol Configuration Register 1 (PCR1)

26.5.2.67.3 Protocol Configuration Register 2 (PCR2)

Base + 0x00A4 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	minislot_after_action_point								number_of_static_slots							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-70. Protocol Configuration Register 2 (PCR2)

26.5.2.67.4 Protocol Configuration Register 3 (PCR3)

Base + 0x00A6 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	wakeup_symbol_rx_low						minislot_action_point_offset[4:0]						coldstart_attempts			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-71. Protocol Configuration Register 3 (PCR3)

26.5.2.67.5 Protocol Configuration Register 4 (PCR4)

Base + 0x00A8 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	cas_rx_low_max								wakeup_symbol_rx_window							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-72. Protocol Configuration Register 4 (PCR4)

26.5.2.67.6 Protocol Configuration Register 5 (PCR5)

Base + 0x00AA — Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	tss_transmitter				wakeup_symbol_tx_low				wakeup_symbol_rx_idle							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-73. Protocol Configuration Register 5 (PCR5)

26.5.2.67.7 Protocol Configuration Register 6 (PCR6)

Base + 0x00AC — Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	symbol_window_after_action_point							macro_initial_offset_a							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-74. Protocol Configuration Register 6 (PCR6)

26.5.2.67.8 Protocol Configuration Register 7 (PCR7)

Base + 0x00AE — Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	decoding_correction_b								micro_per_macro_nom_half							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-75. Protocol Configuration Register 7 (PCR7)

26.5.2.67.9 Protocol Configuration Register 8 (PCR8)

Base + 0x00B0 — Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	max_without_clock_correction_fatal				max_without_clock_correction_passive				wakeup_symbol_tx_idle							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-76. Protocol Configuration Register 8 (PCR8)

26.5.2.67.10 Protocol Configuration Register 9 (PCR9)

Base + 0x00B2 — Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	mini slot_ exists	sym bol_ win dow_ exists	offset_correction_out													
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-77. Protocol Configuration Register 9 (PCR9)

26.5.2.67.11 Protocol Configuration Register 10 (PCR10)

Base + 0x00B4 — Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	single_slot_enabled	wake up_channel	macro_per_cycle													
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-78. Protocol Configuration Register 10 (PCR10)

26.5.2.67.12 Protocol Configuration Register 11 (PCR11)

Base + 0x00B6 — Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	key_slot_used_for_startup	key_slot_used_for_sync	offset_correction_start													
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-79. Protocol Configuration Register 11 (PCR11)

26.5.2.67.13 Protocol Configuration Register 12 (PCR12)

Base + 0x00B8 — Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	allow_passive_to_active				key_slot_header_crc											
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-80. Protocol Configuration Register 12 (PCR12)

26.5.2.67.14 Protocol Configuration Register 13 (PCR13)

Base + 0x00BA — Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	first_minislot_action_point_offset								static_slot_after_action_point							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-81. Protocol Configuration Register 13 (PCR13)

26.5.2.67.15 Protocol Configuration Register 14 (PCR14)

Base + 0x00BC — Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	rate_correction_out												listen_timeout[20:16]			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-82. Protocol Configuration Register 14 (PCR14)

26.5.2.67.16 Protocol Configuration Register 15 (PCR15)

Base + 0x00BE — Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	listen_timeout[15:0]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-83. Protocol Configuration Register 15 (PCR15)

26.5.2.67.17 Protocol Configuration Register 16 (PCR16)

Base + 0x00C0 — Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	macro_initial_offset_b								noise_listen_timeout[24:16]							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-84. Protocol Configuration Register 16 (PCR16)

26.5.2.67.18 Protocol Configuration Register 17 (PCR17)

Base + 0x00C2 — Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	noise_listen_timeout[15:0]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-85. Protocol Configuration Register 17 (PCR17)

26.5.2.67.19 Protocol Configuration Register 18 (PCR18)

Base + 0x00C4 — Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	wakeup_pattern								key_slot_id							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-86. Protocol Configuration Register 18 (PCR18)

26.5.2.67.20 Protocol Configuration Register 19 (PCR19)

Base + 0x00C6 — Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	decoding_correction_a								payload_length_static							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-87. Protocol Configuration Register 19 (PCR19)

26.5.2.67.21 Protocol Configuration Register 20 (PCR20)

Base + 0x00C8 — Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	micro_initial_offset_b								micro_initial_offset_a							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-88. Protocol Configuration Register 20 (PCR20)

26.5.2.67.22 Protocol Configuration Register 21 (PCR21)

Base + 0x00CA — Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	extern_rate_ correction				latest_tx											
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-89. Protocol Configuration Register 21 (PCR21)

26.5.2.67.23 Protocol Configuration Register 22 (PCR22)

Base + 0x00CC — Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	R*	comp_accepted_startup_range_a											micro_per_cycle[19:16			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-90. Protocol Configuration Register 22 (PCR22)

26.5.2.67.24 Protocol Configuration Register 23 (PCR23)

Base + 0x00CE — Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	micro_per_cycle[15:0]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-91. Protocol Configuration Register 23 (PCR23)

26.5.2.67.25 Protocol Configuration Register 24 (PCR24)

Base + 0x00D0 — Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	cluster_drift_damping				max_payload_length_dynamic								micro_per_cycle_min [19:16]			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-92. Protocol Configuration Register 24 (PCR24)

26.5.2.67.26 Protocol Configuration Register 25 (PCR25)

Base + 0x00D2 — Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	micro_per_cycle_min[15:0]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-93. Protocol Configuration Register 25 (PCR25)

26.5.2.67.27 Protocol Configuration Register 26 (PCR26)

Base + 0x00D4 — Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	allow_halt_due_to_clock	comp_accepted_startup_range_b											micro_per_cycle_max [19:16]			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-94. Protocol Configuration Register 26 (PCR26)

26.5.2.67.28 Protocol Configuration Register 27 (PCR27)

Base + 0x00D6 — Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	micro_per_cycle_max[15:0]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-95. Protocol Configuration Register 27 (PCR27)

26.5.2.67.29 Protocol Configuration Register 28 (PCR28)

Base + 0x00D8 — Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	dynamic_slot_idle_phase	macro_after_offset_correction														
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-96. Protocol Configuration Register 28 (PCR28)

26.5.2.67.30 Protocol Configuration Register 29 (PCR29)

Base + 0x00DA — Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	extern_offset_correction				minislots_max											
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-97. Protocol Configuration Register 29 (PCR29)

26.5.2.67.31 Protocol Configuration Register 30 (PCR30)

Base + 0x00DC — Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	sync_node_max			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-98. Protocol Configuration Register 30 (PCR30)

26.5.2.68 Message Buffer Configuration, Control, Status Registers (MBCCSR*n*)

Base + 0x0100 (MBCCSR0)
Base + 0x0108 (MBCCSR1)
...
Base + 0x04F8 (MBCCSR127)

Write: MCM, MBT, MTD: *POC:config* or MB_DIS
CMT: MB_LCK or MB_DIS
EDT, LCKT, MBIE, MBIF: Normal Mode

Additional Reset: CMT, DUP, DVAL, MBIF: Message Buffer Disable

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	MCM	MBT	MTD	CMT	0	0	MBIE	0	0	0	DUP	DVAL	EDS	LCKS	MBIF
W					rwm	EDT	LCKT									w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-99. Message Buffer Configuration, Control, Status Registers (MBCCSR*n*)

The content of these registers comprises message buffer configuration data, message buffer control data, message buffer status information, and message buffer interrupt flags. A detailed description of all flags can be found in Section 26.6.6, Individual Message Buffer Functional Description.

If the application writes 1 to the EDT bit, no write access to the other register bits is performed.

If the application writes 0 to the EDT bit and 1 to the LCKT bit, no write access to the other bits is performed.

Table 26-79. MBCCSR*n* Field Descriptions

Field	Description
Message Buffer Configuration	
MCM	**Message Buffer Commit Mode** — This bit configures the commit mode of a double buffered message buffer. 0 Streaming commit mode. 1 Immediate commit mode.
MBT	**Message Buffer Type** — This bit configures the buffering type of a transmit message buffer. 0 Single buffered message buffer. 1 Double buffered message buffer.
MTD	**Message Buffer Transfer Direction** — This bit configures the transfer direction of a message buffer. 0 Receive message buffer. 1 Transmit message buffer.
Message Buffer Control	
CMT	**Commit for Transmission** — This bit indicates if the transmit message buffer data are ready for transmission. 0 Message buffer data not ready for transmission. 1 Message buffer data ready for transmission.
EDT	**Enable/Disable Trigger** — If the application writes 1 to this bit, a message buffer enable or disable is triggered, depending on the current value EDS status bit is 0. 0 No effect. 1 Message buffer enable or disable is triggered.
LCKT	**Lock/Unlock Trigger** — If the application writes 1 to this bit, a message buffer lock or unlock is triggered, depending on the current value of the LCKS status bit. 0 No effect. 1 Message buffer lock or unlock is triggered.
MBIE	**Message Buffer Interrupt Enable** — This control bit defines whether the message buffer will generate an interrupt request when its MBIF flag is set. 0 Interrupt request generation disabled. 1 Interrupt request generation enabled.
Message Buffer Status	
DUP	**Data Updated** — This status bit indicates whether the frame header in the message buffer header field and the data in the message buffer data field were updated after a frame reception. 0 Frame Header and Message buffer data field not updated. 1 Frame Header and Message buffer data field updated.
DVAL	**Data Valid** — For receive message buffers this status bit indicates whether the message buffer data field contains valid frame data. For transmit message buffers the status bit indicates if a message is transferred again due to the state transmission mode of the message buffer. 0 receive message buffer contains no valid frame data / message is transmitted for the first time. 1 receive message buffer contains valid frame data / message will be transferred again.
EDS	**Enable/Disable Status** — This status bit indicates whether the message buffer is enabled or disabled. 0 Message buffer is disabled. 1 Message buffer is enabled.
LCKS	**Lock Status** — This status bit indicates the current lock status of the message buffer. 0 Message buffer is not locked by the application. 1 Message buffer is locked by the application.
MBIF	**Message Buffer Interrupt Flag** — This flag is set when the slot status field of the message buffer was updated after frame transmission or reception, or when a transmit message buffer was just enabled by the application. 0 No such event. 1 Slot status field updated or transmit message buffer just enabled.

26.5.2.69 Message Buffer Cycle Counter Filter Registers (MBCCFR*n*)

Base + 0x0102 (MBCCFR0)
Base + 0x010A (MBCCFR1)
...
Base + 0x04FA (MBCCFR127)

Write: *POC:config* or MB_DIS

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	MTM	CHA	CHB	CCFE	CCFMSK						CCFVAL					
Reset	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

Figure 26-100. Message Buffer Cycle Counter Filter Registers (MBCCFR*n*)

This register contains message buffer configuration data for the transmission mode, the channel assignment, and for the cycle counter filtering. For detailed information on cycle counter filtering, refer to Section 26.6.7.1, Message Buffer Cycle Counter Filtering.

Table 26-80. MBCCFR*n* Field Descriptions

Field	Description
MTM	**Message Buffer Transmission Mode** — This control bit applies only to transmit message buffers and defines the transmission mode. 0 Event transmission mode. 1 State transmission mode.
CHA CHB	**Channel Assignment** — These control bits define the channel assignment and control the receive and transmit behavior of the message buffer according to **Table 26-81**.
CCFE	**Cycle Counter Filtering Enable** — This control bit is used to enable and disable the cycle counter filtering. 0 Cycle counter filtering disabled. 1 Cycle counter filtering enabled.
CCFMSK	Cycle Counter Filtering Mask — This field defines the filter mask for the cycle counter filtering.
CCFVAL	**Cycle Counter Filtering Value** — This field defines the filter value for the cycle counter filtering.

Table 26-81. Channel Assignment Description

CHA	CHB	Transmit Message Buffer		Receive Message Buffer	
		static segment	dynamic segment	static segment	dynamic segment
1	1	transmit on both channel A and channel B	transmit on channel A only	store first valid frame received on either channel A or channel B	store first valid frame received on channel A, ignore channel B
0	1	transmit on channel B	transmit on channel B	store first valid frame received on channel B	store first valid frame received on channel B
1	0	transmit on channel A	transmit on channel A	store first valid frame received on channel A	store first valid frame received on channel A
0	0	no frame transmission	no frame transmission	no frame stored	no frame stored

NOTE

If at least one message buffer assigned to a certain slot is assigned to both channels, then all message buffers assigned to this slot have to be assigned to both channels. Otherwise, the message buffer configuration is illegal and the result of the message buffer search is not defined.

26.5.2.70 Message Buffer Frame ID Registers (MBFIDR*n*)

Base + 0x0104 (MBFIDR0)
Base + 0x010C (MBFIDR1)
...
Base + 0x04FC (MBFIDR127)

Write: *POC:config* or MB_DIS

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	FID										
W																
Reset	0	0	0	0	0	-	-	-	-	-	-	-	-	-	-	-

Figure 26-101. Message Buffer Frame ID Registers (MBFIDR*n*)

Table 26-82. MBFIDR*n* Field Descriptions

Field	Description
FID	**Frame ID** — The semantic of this field depends on the message buffer transfer type. • *Receive Message Buffer:* This field is used as a filter value to determine if the message buffer is used for reception of a message received in a slot with the slot ID equal to FID. • *Transmit Message Buffer:* This field is used to determine the slot in which the message in this message buffer should be transmitted.

26.5.2.71 Message Buffer Index Registers (MBIDXR*n*)

Base + 0x0106 (MBIDXR0)
Base + 0x010E (MBIDXR1)
...
Base + 0x04FE (MBIDXR127)

Write: *POC:config* or MB_DIS

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	MBIDX							
W																
Reset	0	0	0	0	0	0	0	0	-	-	-	-	-	-	-	-

Figure 26-102. Message Buffer Index Registers (MBIDXR*n*)

Table 26-83. MBIDXR*n* Field Descriptions

Field	Description
MBIDX	**Message Buffer Index** — This field provides the index of the message buffer header field of the physical message buffer that is currently associated with this message buffer. The application writes the index of the initially associated message buffer header field into this register. The controller updates this register after frame reception or transmission.

NOTE

If a message buffer is assigned to the last slot in a FlexyRay communication cycle and a system memory access timeout or illegal address access occurs during the system memory access in this slot, it is possible that for all future communication required:

- No slot status information will be written,
- The message buffer status will not be updated, and
- No message frames will be received. If this happens, several message buffers can never be locked by the application.

However, if this occurs, either the System Bus Communication Failure Error Flag (SBCF_EF) or the Illegal System Bus Address Error Flag (ILSA_EF) will be set in the Controller Host Interface Error Flag Register (CHIERFR).

The FlexRay module and the system memory subsystem should be configured to avoid the occurrence of system memory access timeouts and illegal address accesses. In case, one of the error flags CHIERFR[SBCF_EF] or CHIERFR[ILSA_EF] is set, the application should stop the FlexRay controller via a FREEZE or HALT command and subsequently restart the controller.

26.6 Functional Description

This section provides a detailed description of the functionality implemented in the controller.

26.6.1 Message Buffer Concept

The controller uses a data structure called *message buffer* to store frame data, configuration, control, and status data. Each message buffer consists of two parts, the *message buffer control data* and the *physical message buffer.* The message buffer control data are located in dedicated registers. The structure of the message buffer control data depends on the message buffer type and is described in Section 26.6.3, Message Buffer Types. The physical message buffer is located in the FlexRay memory and is described in Section 26.6.2, Physical Message Buffer.

26.6.2 Physical Message Buffer

All FlexRay messages and related frame and slot status information of received frames and of frames to be transmitted to the FlexRay bus are stored in data structures called *physical message buffers*. The physical message buffers are located in the FlexRay memory.The structure of a physical message buffer is depicted in Figure 26-103.

A physical message buffer consists of two fields, the *message buffer header field* and the *message buffer data field*. The message buffer header field contains the *frame header*, the *data field offset*, and the *slot status*.The message buffer data field contains the *frame data*.

The connection between the two fields is established by the *data field offset*.

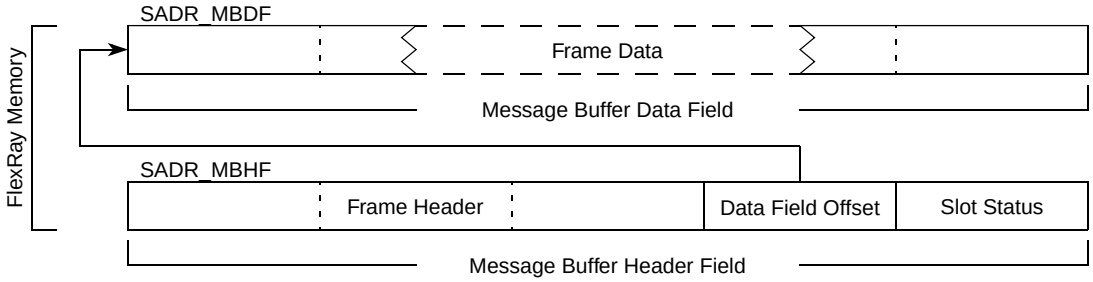


Figure 26-103. Physical Message Buffer Structure

26.6.2.1 Message Buffer Header Field

The message buffer header field is a contiguous region in the FlexRay memory and occupies ten bytes. It contains the frame header, the data field offset, and the slot status. Its structure is shown in Figure 26-103. The physical start address *SADR_MBHF* of the message buffer header field must be 16-bit aligned.

26.6.2.1.1 Frame Header

The frame header occupies the first six bytes in the message buffer header field. It contains all FlexRay frame header related information according to the *FlexRay Communications System Protocol*

Specification, Version 2.1 Rev A. A detailed description of the usage and the content of the frame header is provided in Section 26.6.5.2.1, Frame Header Description.

26.6.2.1.2 Data Field Offset

The data field offset follows the frame header in the message buffer data field and occupies two bytes. It contains the offset of the corresponding message buffer data field with respect to the controller FlexRay memory base address as provided by SMBA field in the System Memory Base Address Register (SYMBADR). The data field offset is used to determine the start address *SADR_MBDF* of the corresponding message buffer data field in the FlexRay memory according to Equation 26-2.

SADR_MBDF = [Data Field Offset] + SMBA ***Eqn. 26-2***

26.6.2.1.3 Slot Status

The slot status occupies the last two bytes of the message buffer header field. It provides the slot and frame status related information according to the *FlexRay Communications System Protocol Specification, Version 2.1 Rev A*. A detailed description of the content and usage of the slot status is provided in Section 26.6.5.2.3, Slot Status Description.

26.6.2.2 Message Buffer Data Field

The message buffer data field is a contiguous area of 2-byte entities. This field contains the frame payload data, or a part of it, of the frame to be transmitted to or received from the FlexRay bus. The minimum length of this field depends on the specific message buffer configuration and is specified in the message buffer descriptions given in Section 26.6.3, Message Buffer Types.

26.6.3 Message Buffer Types

The controller provides three different types of message buffers.

- Individual Message Buffers
- Receive Shadow Buffers
- Receive FIFO Buffers

For each message buffer type the structure of the physical message buffer is identical. The message buffer types differ only in the structure and content of message buffer control data, which control the related physical message buffer. The message buffer control data are described in the following sections.

26.6.3.1 Individual Message Buffers

The individual message buffers are used for all types of frame transmission and for dedicated frame reception based on individual filter settings for each message buffer. The controller supports three types of individual message buffers, which are described in Section 26.6.6, Individual Message Buffer Functional Description.

Each individual message buffer consists of two parts, the physical message buffer, which is located in the FlexRay memory, and the message buffer control data, which are located in dedicated registers. The structure of an individual message buffer is given in Figure 26-104.

Each individual message buffer has a message buffer number *n* assigned, which determines the set of message buffer control registers associated to this individual message buffer. The individual message buffer with message buffer number *n* is controlled by the registers MBCCSR*n*, MBCCFR*n*, MBFIDR*n*, and MBIDXR*n*.

The connection between the message buffer control registers and the physical message buffer is established by the message buffer index field MBIDX in the Message Buffer Index Registers (MBIDXRn). The start address SADR_MBHF of the related message buffer header field in the FlexRay memory is determined according to Equation 26-3.

$$\text{SADR\_MBHF} = (\text{MBIDXR}n[\text{MBIDX}] * 10) + \text{SMBA} \qquad \textit{Eqn. 26-3}$$

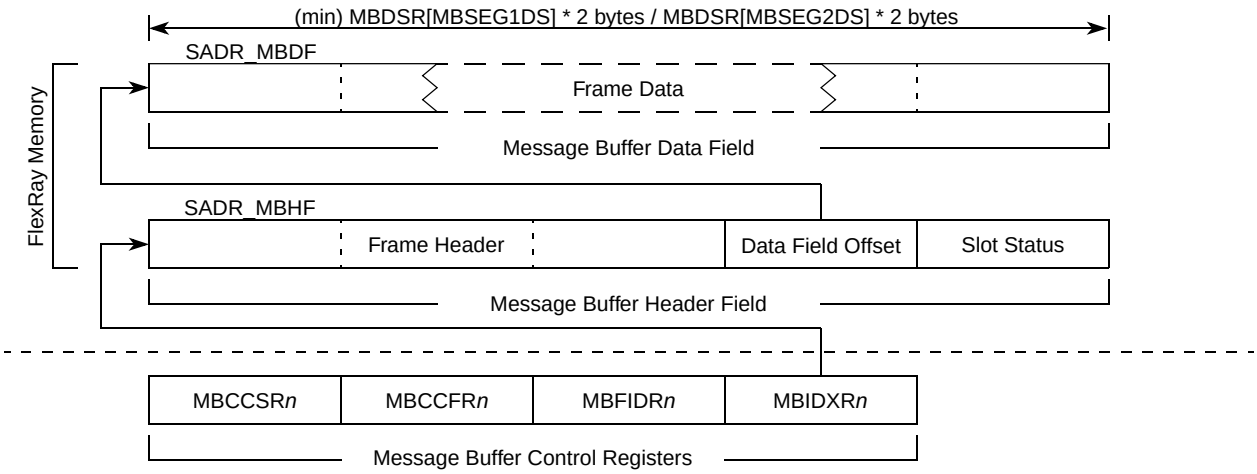


Figure 26-104. Individual Message Buffer Structure

26.6.3.1.1 Individual Message Buffer Segments

The set of the individual message buffers can be split up into two message buffer segments using the Message Buffer Segment Size and Utilization Register (MBSSUTR). All individual message buffers with a message buffer number $n \leq$ MBSSUTR[LAST_MB_SEG1] belong to the first message buffer segment. All individual message buffers with a message buffer number $n >$ MBSSUTR[LAST_MB_SEG1] belong to the second message buffer segment. The following rules apply to the length of the message buffer data field:

- all physical message buffers associated to individual message buffers that belong to the same message buffer segment must have message buffer data fields of the same length
- the minimum length of the message buffer data field for individual message buffers in the first message buffer segment is 2 * MBDSR[MBSEG1DS] bytes
- the minimum length of the message buffer data field for individual message buffers assigned to the second segment is 2 * MBDSR[MBSEG2DS] bytes.

26.6.3.2 Receive Shadow Buffers

The receive shadow buffers are required for the frame reception process for individual message buffers. The controller provides four receive shadow buffers, one receive shadow buffer per channel and per message buffer segment.

Each receive shadow buffer consists of two parts, the physical message buffer located in the FlexRay memory and the receive shadow buffer control registers located in dedicated registers. The structure of a receive shadow buffer is shown in Figure 26-105. The four internal shadow buffer control registers can be accessed by the Receive Shadow Buffer Index Register (RSBIR).

The connection between the receive shadow buffer control register and the physical message buffer for the selected receive shadow buffer is established by the receive shadow buffer index field RSBIDX in the Receive Shadow Buffer Index Register (RSBIR). The start address SADR_MBHF of the related message buffer header field in the FlexRay memory is determined according to Equation 26-4.

$$SADR\_MBHF = (RSBIR[RSBIDX] * 10) + SMBA \qquad \textit{Eqn. 26-4}$$

The length required for the message buffer data field depends on the message buffer segment that the receive shadow buffer is assigned to. For the receive shadow buffers assigned to the first message buffer segment, the length must be the same as for the individual message buffers assigned to the first message buffer segment. For the receive shadow buffers assigned to the second message buffer segment, the length must be the same as for the individual message buffers assigned to the second message buffer segment. The receive shadow buffer assignment is described in Receive Shadow Buffer Index Register (RSBIR).

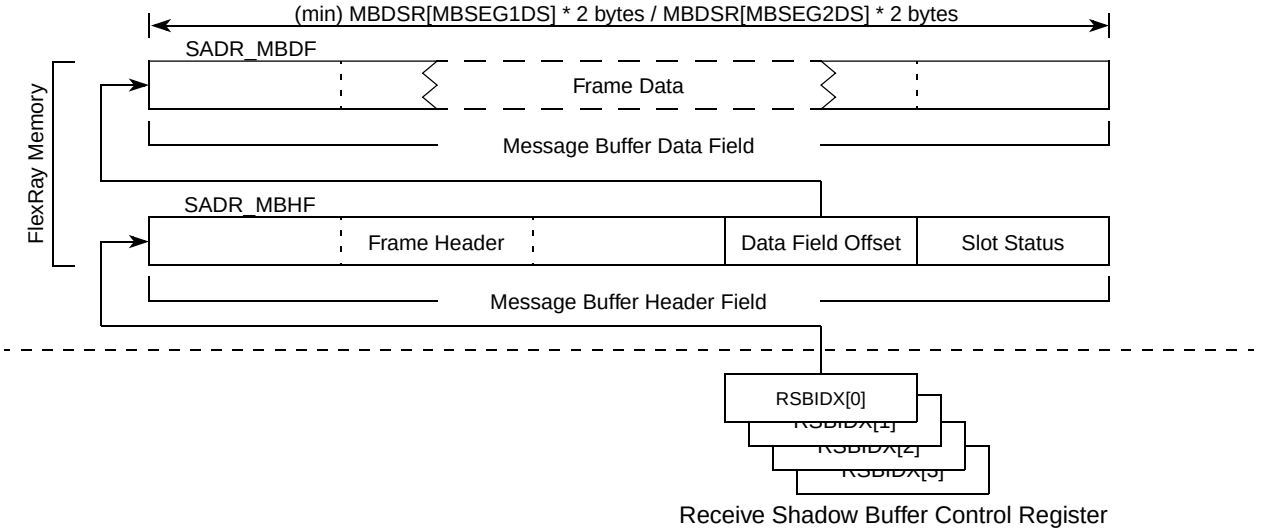


Figure 26-105. Receive Shadow Buffer Structure

26.6.3.3 Receive FIFO

The receive FIFO implements a frame reception system based on the FIFO concept. The controller provides two independent receive FIFOs, one per channel.

A receive FIFO consists of a set of physical message buffers in the FlexRay memory and a set of receive FIFO control registers located in dedicated registers. The structure of a receive FIFO is given in Figure 26-106.

The connection between the receive FIFO control registers and the set of physical message buffers is established by the Receive FIFO Start Index Register (RFSIR), the Receive FIFO Depth and Size Register (RFDSR), and the Receive FIFO A Read Index Register (RFARIR) / Receive FIFO B Read Index Register (RFBRIR). The system memory base address SMBA is defined by the system memory base address register selected by the FIFO address mode bit MCR[FAM].

The start byte address SADR_MBHF[1] of the first message buffer header field that belongs to the receive FIFO in the FlexRay memory is determined according to Equation 26-5.

$$\text{SADR\_MBHF[1]} = (10 * \text{RFSIR[SIDX]}) + \text{SMBA} \qquad \textit{Eqn. 26-5}$$

The start byte address SADR_MBHF[n] of the last message buffer header field that belongs to the receive FIFO in the FlexRay memory is determined according to Equation 26-6.

$$\text{SADR\_MBHF[n]} = (10 * (\text{RFSIR[SIDX]} + \text{RFDSR[FIFO\_DEPTH]})) + \text{SMBA} \qquad \textit{Eqn. 26-6}$$

NOTE

All message buffer header fields assigned to a receive FIFO must be a contiguous region.

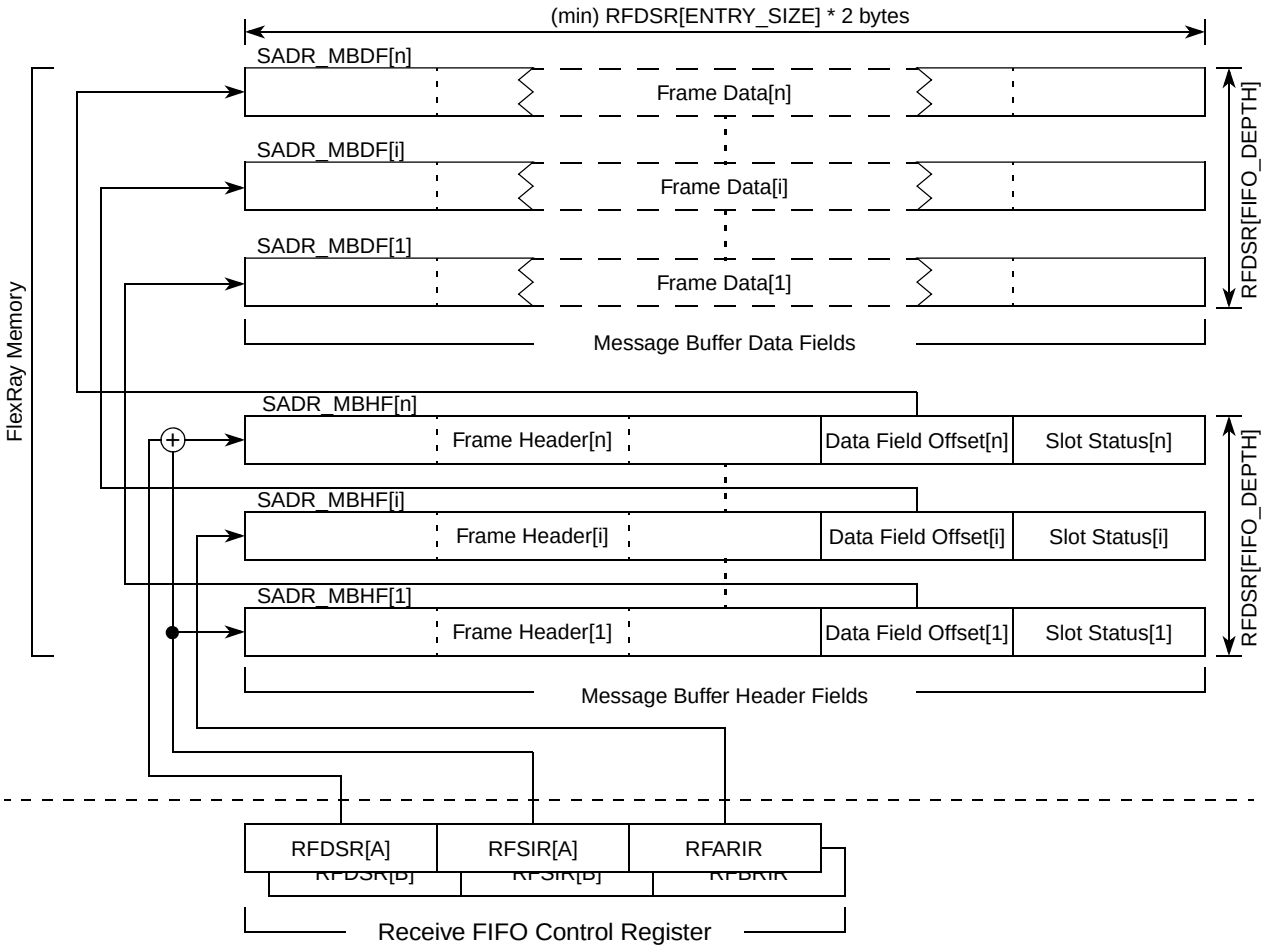


Figure 26-106. Receive FIFO Structure

26.6.3.4 Message Buffer Configuration and Control Data

This section describes the configuration and control data for each message buffer type.

26.6.3.4.1 Individual Message Buffer Configuration Data

Before an individual message buffer can be used for transmission or reception, it must be configured. There is a set of common configuration parameters that applies to all individual message buffers and a set of configuration parameters that applies to each message buffer individually.

Common Configuration Data

The set of common configuration data for individual message buffers is located in the following registers.

- Message Buffer Data Size Register (MBDSR)
  The MBSEG2DS and MBSEG1DS fields define the minimum length of the message buffer data field with respect to the message buffer segment.
- Message Buffer Segment Size and Utilization Register (MBSSUTR)
  The LAST_MB_SEG1 and LAST_MB_UTIL fields define the segmentation of the individual

message buffers and the number of individual message buffers that are used. For more details, see Section 26.6.3.1.1, Individual Message Buffer Segments.

Specific Configuration Data

The set of message buffer specific configuration data for individual message buffers is located in the following registers.

- Message Buffer Configuration, Control, Status Registers (MBCCSRn)
  The MCM, MBT, MTD bits configure the message buffer type.
- Message Buffer Cycle Counter Filter Registers (MBCCFRn)
  The MTM, CHA, CHB bits configure the transmission mode and the channel assignment. The CCFE, CCFMSK, and CCFVAL bits and fields configure the cycle counter filter.
- Message Buffer Frame ID Registers (MBFIDRn)
  For a transmit message buffer, the FID field is used to determine the slot in which the message in this message buffer will be transmitted.
- Message Buffer Index Registers (MBIDXRn)
  This MBIDX field provides the index of the message buffer header field of the physical message buffer that is currently associated with this message buffer.

26.6.3.5 Individual Message Buffer Control Data

During normal operation, each individual message buffer can be controlled by the control and trigger bits CMT, LCKT, EDT, and MBIE in the Message Buffer Configuration, Control, Status Registers (MBCCSRn).

26.6.3.6 Receive Shadow Buffer Configuration Data

Before frame reception into the individual message buffers can be performed, the receive shadow buffers must be configured. The configuration data are provided by the Receive Shadow Buffer Index Register (RSBIR). For each receive shadow buffer, the application provides the message buffer header index. When the protocol is in the *POC:normal active* or *POC:normal passive* state, the receive shadow buffers are under full controller control.

26.6.3.7 Receive FIFO Control and Configuration Data

This section describes the configuration and control data for the two receive FIFOs.

26.6.3.7.1 Receive FIFO Configuration Data

The controller provides two functional independent receive FIFOs, one per channel. The FIFOs have a common subset of configuration data:

- Receive FIFO System Memory Base Address Register (RFSYMBADR)
- Receive FIFO Periodic Timer Register (RFPTR)

Each FIFO has its own set of configuration data. The configuration data are located in the following registers:

- Receive FIFO Watermark and Selection Register (RFWMSR)
- Receive FIFO Start Index Register (RFSIR)
- Receive FIFO Depth and Size Register (RFDSR)
- Receive FIFO Message ID Acceptance Filter Value Register (RFMIDAFVR)
- Receive FIFO Message ID Acceptance Filter Mask Register (RFMIAFMR)
- Receive FIFO Frame ID Rejection Filter Value Register (RFFIDRFVR)
- Receive FIFO Frame ID Rejection Filter Mask Register (RFFIDRFMR)
- Receive FIFO Range Filter Configuration Register (RFRFCFR)

26.6.3.7.2 Receive FIFO Control Data

The application can access the FIFOs at any time using the control bits in the following registers:

- Global Interrupt Flag and Enable Register (GIFER)
- Receive FIFO Fill Level and POP Count Register (RFFLPCR)

26.6.3.7.3 Receive FIFO Status Data

The current status of the receive fifo is provided in the following register:

- Global Interrupt Flag and Enable Register (GIFER)
- Receive FIFO A Read Index Register (RFARIR)
- Receive FIFO B Read Index Register (RFBRIR)
- Receive FIFO Fill Level and POP Count Register (RFFLPCR)

26.6.4 FlexRay Memory Layout

The controller supports a wide range of possible layouts for the FlexRay memory. Two basic layout modes can be selected by the FIFO address mode bit MCR[FAM].

26.6.4.1 FlexRay Memory Layout (MCR[FAM] = 0)

Figure 26-107 shows an example layout for the FIFO address mode MCR[FAM] = 0. In this mode, the following set of rules applies to the layout of the FlexRay memory:

- The FlexRay memory is one contiguous region.
- The FlexRay memory size is maximum 64 Kbytes.
- The FlexRay memory starts at a 16 byte boundary

The FlexRay memory contains three areas: the *message buffer header area*, the *message buffer data area*, and the *sync frame table area*.

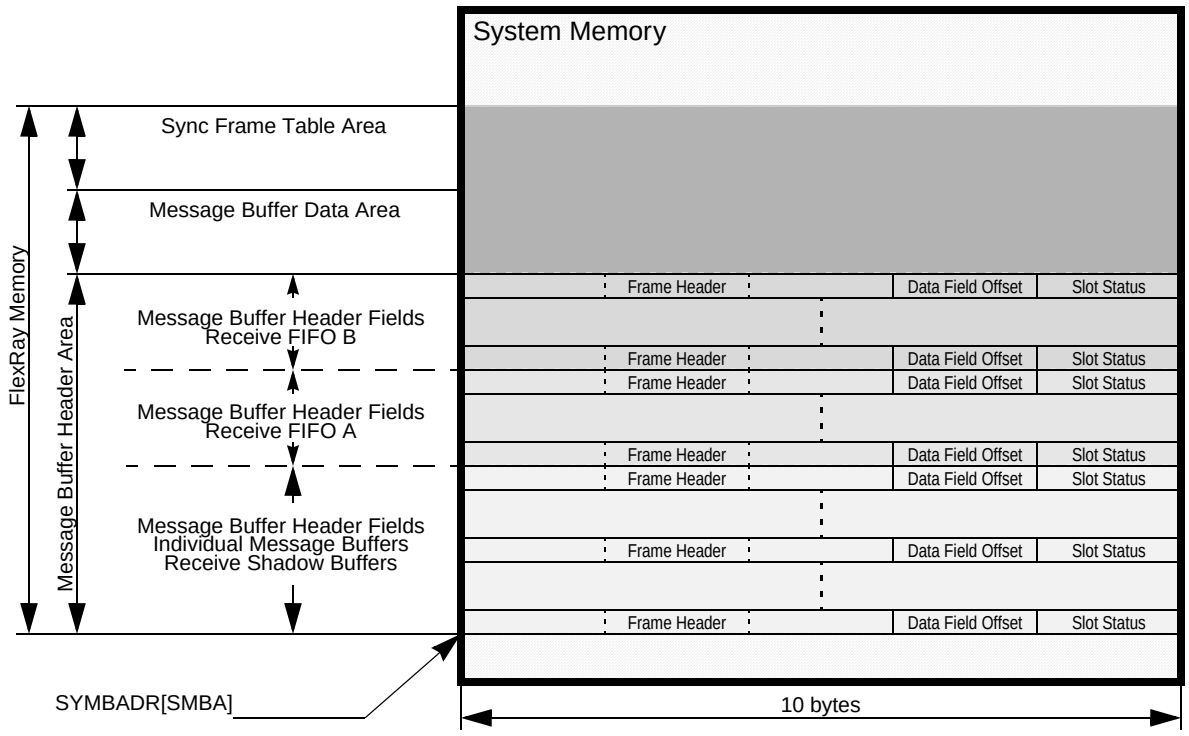


Figure 26-107. Example of FlexRay Memory Layout (MCR[FAM] = 0)

26.6.4.2 FlexRay Memory Layout (MCR[FAM] = 1)

Figure 26-108 shows an example layout for the FIFO address mode MCR[FAM] = 1. The following set of rules applies to the layout of the FlexRay memory:

- The FlexRay memory consists of two contiguous regions.
- The size of each region is maximum 64 Kbytes.
- Each region start at a 16 byte boundary.

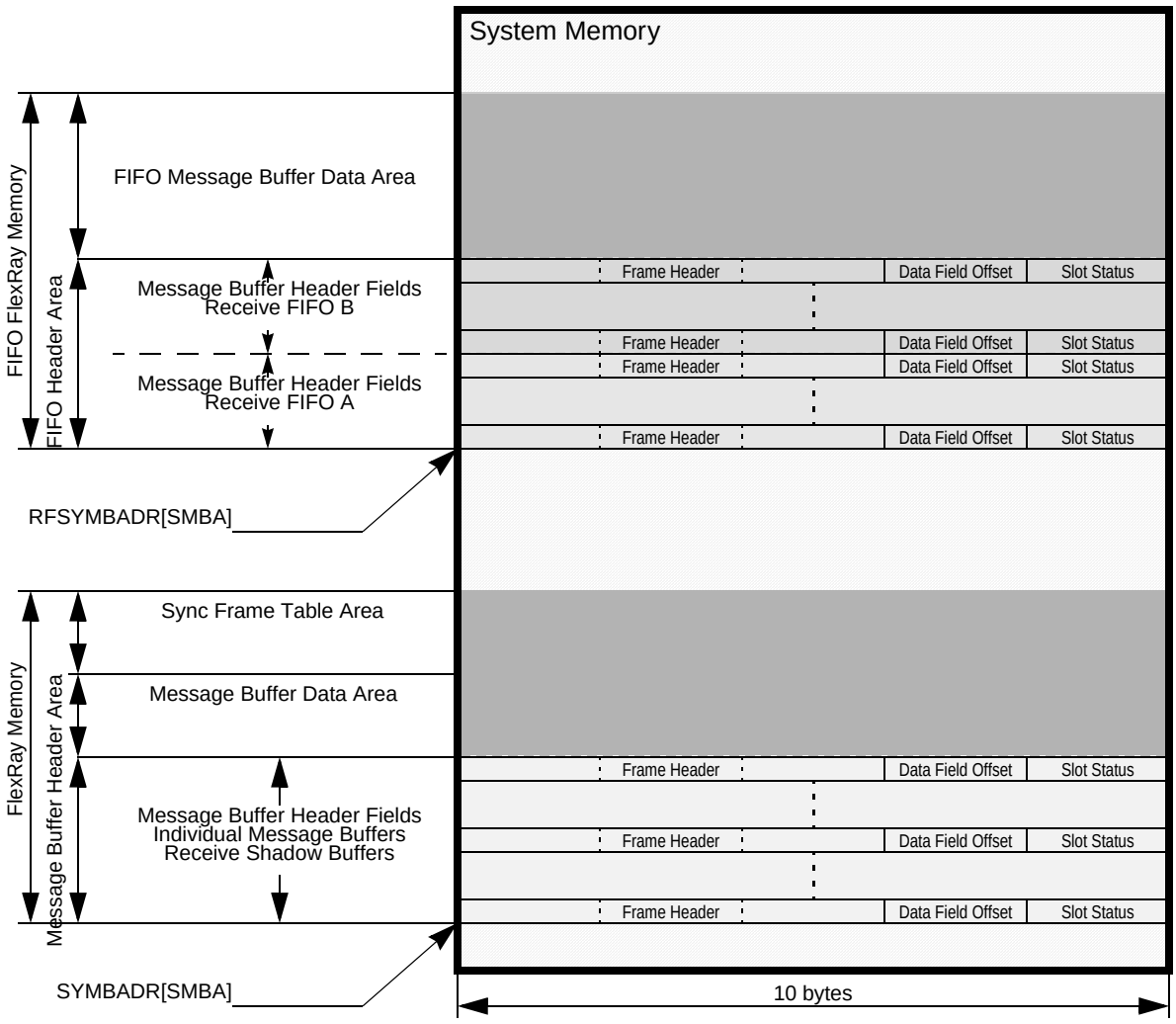


Figure 26-108. Example of FlexRay Memory Layout (MCR[FAM] = 1)

26.6.4.3 Message Buffer Header Area (MCR[FAM] = 0)

The message buffer header area contains all message buffer header fields of the physical message buffers for all message buffer types. The following rules apply to the message buffer header fields for the three type of message buffers.

1. The start byte address SADR_MBHF of each message buffer header field for *individual message buffers* and *receive shadow buffers* must fulfill Equation 26-7.

$$SADR\_MBHF = (i * 10) + SYMBADR[SMBA];\ (0 \leq i < 256) \qquad \textit{Eqn. 26-7}$$

2. The start byte address SADR_MBHF of each message buffer header field for the *FIFO* must fulfill Equation 26-8.

$$SADR\_MBHF = (i * 10) + SYMDARD[SMBA];\ (0 \leq i < 1024) \qquad \textit{Eqn. 26-8}$$

$$SADR\_MBHF = (i * 10) + SYMBADR[SMBA]; (0 \leq i < 1024) \quad \textit{Eqn. 26-9}$$

3. The message buffer header fields for each FIFO have to be a contiguous area.

26.6.4.4 Message Buffer Header Area (MCR[FAM] = 1)

The message buffer header area contains all message buffer header fields of the physical message buffers for the individual message buffers and receiver shadow buffers. The following rules apply to the message buffer header fields for the two type of message buffers.

1. The start address SADR_MBHF of each message buffer header field for *individual message buffers* and *receive shadow buffers* must fulfill Equation 26-10.

$$SADR\_MBHF = (i * 10) + SYMBADR[SMBA]; (0 \leq i < 256) \quad \textit{Eqn. 26-10}$$

26.6.4.5 FIFO Message Buffer Header Area (MCR[FAM] = 1)

The FIFO message buffer header area contains all message buffer header fields of the physical message buffers for the FIFO. The following rules apply to the FIFO message buffer header fields.

1. The start byte address SADR_MBHF of each message buffer header field for the *FIFO* must fulfill Equation 26-11.

$$SADR\_MBHF = (i * 10) + RFSYMBADR[SMBA]; (0 \leq i < 1024) \quad \textit{Eqn. 26-11}$$

2. The message buffer header fields for each FIFO have to be a contiguous area.

26.6.4.6 Message Buffer Data Area

The message buffer data area contains all the message buffer data fields of the physical message buffers. Each message buffer data field must start at a 16-bit boundary.

26.6.4.7 Sync Frame Table Area

The sync frame table area is used to provide a copy of the internal sync frame tables for application access. Refer to Section 26.6.12, Sync Frame ID and Sync Frame Deviation Tables, for the description of the sync frame table area.

26.6.5 Physical Message Buffer Description

This section provides a detailed description of the usage and the content of the two parts of a physical message buffer, the message buffer header field and the message buffer data field.

26.6.5.1 Message Buffer Protection and Data Consistency

The physical message buffers are located in the FlexRay memory. The controller provides no means to protect the FlexRay memory from uncontrolled or illegal host or other client write access. To ensure data consistency of the physical message buffers, the application must follow the write access scheme that is given in the description of each of the physical message buffer fields.

26.6.5.2 Message Buffer Header Field Description

This section provides a detailed description of the usage and content of the message buffer header field. A description of the structure of the message buffer header fields is given in Section 26.6.2.1, Message Buffer Header Field. Each message buffer header field consists of three sections: the frame header section, the data field offset, and the slot status section. For a detailed description of the Data Field Offset, see Section 26.6.2.1.2, Data Field Offset.

26.6.5.2.1 Frame Header Description

Frame Header Content

The semantic and content of the frame header section depends on the message buffer type.

For individual receive message buffers and receive FIFOs, the frame header receives the frame header data of the *first valid frame* received on the assigned channels.

For receive shadow buffers, the frame header receives the frame header data of the current frame received regardless of whether the frame is valid or not.

For transmit message buffers, the application writes the frame header of the frame to be transmitted into this location. The frame header will be read out when the frame is transferred to the FlexRay bus.

The structure of the frame header in the message buffer header field for receive message buffers and the receive FIFO is given in Figure 26-109. A detailed description is given in Table 26-85.

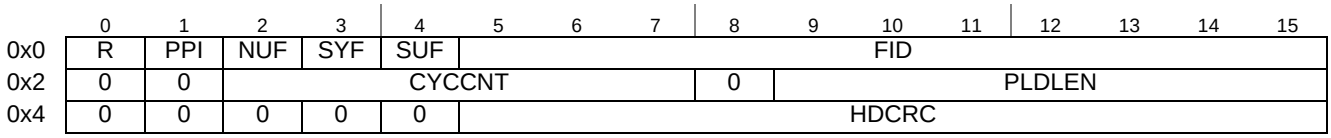


Figure 26-109. Frame Header Structure (Receive Message Buffer and Receive FIFO)

The structure of the frame header in the message buffer header field for transmit message buffers is given in Figure 26-110. A detailed description is given in Table 26-86. The checks that will be performed are described in Frame Header Checks.

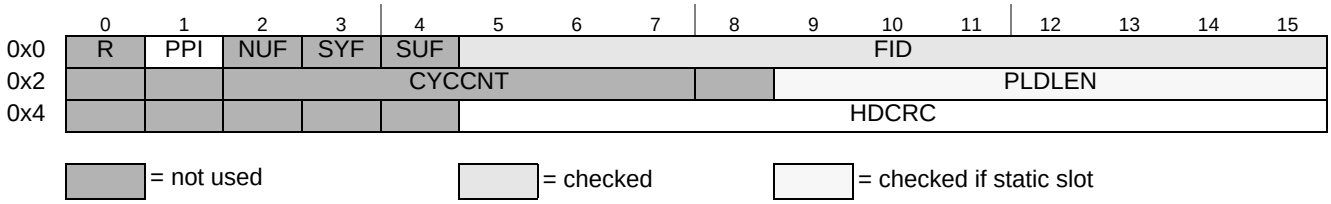


Figure 26-110. Frame Header Structure (Transmit Message Buffer)

The structure of the frame header in the message buffer header field for transmit message buffers assigned to key slot is given in Figure 26-111.

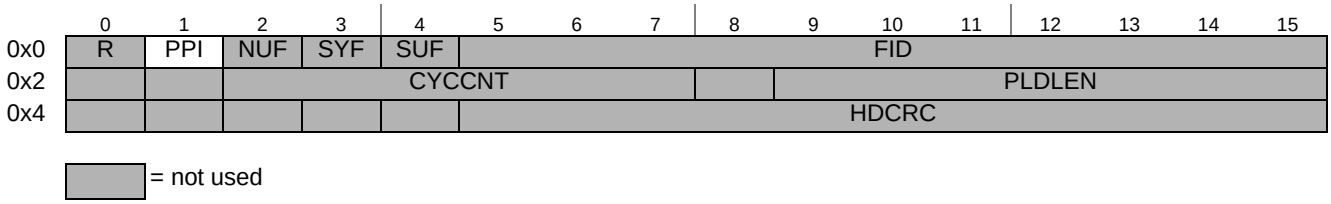


Figure 26-111. Frame Header Structure (Transmit Message Buffer for Key Slot)

Frame Header Access

The frame header is located in the FlexRay memory. To ensure data consistency, the application must follow the write access scheme described below.

For receive message buffers, receive shadow buffers, and receive FIFOs, the application must not write to the frame header field.

For transmit message buffers, the application must follow the write access restrictions given in Table 26-84. This table shows the condition under which the application can write to the frame header entries without corrupting the FlexRay message transmission.

Table 26-84. Frame Header Write Access Constraints (Transmit Message Buffer)

Field	Single Buffered		Double Buffered			
	Static Segment	Dynamic Segment	Static Segment		Dynamic Segment	
			Commit Side	Transmit Side	Commit Side	Transmit Side
FID	*POC:config* or MB_DIS					
PPI, PLDLEN, HDCRC		MB_LCK	*POC:config* or MB_DIS or		MB_LCK	

Frame Header Checks

As shown in Figure 26-110 and Figure 26-111 not all fields in the message buffer frame header are used for transmission. Some fields in the message buffer frame header are ignored, some are used for transmission, and some of them are checked for correct values. All checks that will be performed are described below.

For message buffers assigned to the key slot, no checks will be performed.

The value of the FID field must be equal to the value of the corresponding Message Buffer Frame ID Registers (MBFIDRn). If the controller detects a mismatch while transmitting the frame header, it will set the frame ID error flag FID_EF in the CHI Error Flag Register (CHIERFR). The value of the FID field will be ignored and replaced by the value provided in the Message Buffer Frame ID Registers (MBFIDRn).

For transmit message buffers assigned to the *static* segment, the PLDLEN value must be equal to the value of the payload_length_static field in the Protocol Configuration Register 19 (PCR19). If this is not fulfilled, the static payload length error flag SPL_EF in the CHI Error Flag Register (CHIERFR) is set when the message buffer is under transmission. A syntactically and semantically correct frame is generated with payload_length_static payload words and the payload length field in the transmitted frame header set to payload_length_static.

For transmit message buffers assigned to the *dynamic* segment, the PLDLEN value must be less than or equal to the value of the max_payload_length_dynamic field in the Protocol Configuration Register 24 (PCR24). If this is not fulfilled, the dynamic payload length error flag DPL_EF in the CHI Error Flag Register (CHIERFR) is set when the message buffer is under transmission. A syntactically and semantically correct dynamic frame is generated with PLDLEN payload words and the payload length field in the frame header set to PLDLEN.

Table 26-85. Frame Header Field Descriptions (Receive Message Buffer and Receive FFO)

Field	Description
R	**Reserved Bit** — This is the value of the *Reserved bit* of the received frame stored in the message buffer
PPI	**Payload Preamble Indicator** — This is the value of the *Payload Preamble Indicator* of the received frame stored in the message buffer.
NUF	**Null Frame Indicator** — This is the value of the *Null Frame Indicator* of the received frame stored in the message buffer.
SYF	**Sync Frame Indicator** — This is the value of the *Sync Frame Indicator* of the received frame stored in the message buffer.
SUF	**Startup Frame Indicator** — This is the value of the *Startup Frame Indicator* of the received frame stored in the message buffer.
FID	**Frame ID** — This is the value of the *Frame ID* field of the received frame stored in the message buffer.
CYCCNT	**Cycle Count** — This is the number of the communication cycle in which the frame stored in the message buffer was received.
PLDLEN	**Payload Length** — This is the value of the *Payload Length* field of the received frame stored in the message buffer.
HDCRC	**Header CRC** — This is the value of the *Header CRC* field of the received frame stored in the message buffer.

Table 26-86. Frame Header Field Descriptions (Transmit Message Buffer)

Field	Description
R	**Reserved Bit** — This bit is not used, the value of the *Reserved bit* is generated internally according to *FlexRay Communications System Protocol Specification, Version 2.1 Rev A.*
PPI	**Payload Preamble Indicator** — This bit provides the value of the *Payload Preamble Indicator* for the frame transmitted from the message buffer.
NUF	**Null Frame Indicator** — This bit is not used, the value of the *Null Frame Indicator* is generated internally according to *FlexRay Communications System Protocol Specification, Version 2.1 Rev A.*
SYF	**Sync Frame Indicator** — This bit is not used, the value of the *Sync Frame Indicator* is generated internally according to *FlexRay Communications System Protocol Specification, Version 2.1 Rev A.*
SUF	**Startup Frame Indicator** — This bit is not used, the value of the *Startup Frame Indicator* is generated internally according to *FlexRay Communications System Protocol Specification, Version 2.1 Rev A.*
FID	**Frame ID** — This field is checked as described in Frame Header Checks.
CYCCNT	**Cycle Count** — This field is not used, the value of the transmitted *Cycle Count* field is taken from the internal communication cycle counter.

Table 26-86. Frame Header Field Descriptions (Transmit Message Buffer) (continued)

Field	Description
PLDLEN	**Payload Length** — This field is checked and used as described in Frame Header Checks.
HDCRC	**Header CRC** — This field provides the value of the *Header CRC* field for the frame transmitted from the message buffer.

26.6.5.2.2 Data Field Offset Description

Data Field Offset Content

For a detailed description of the Data Field Offset, see Section 26.6.2.1.2, Data Field Offset.

Data Field Offset Access

The application shall program the Data Field Offset when configuring the message buffers either in the *POC:config* state or when the message buffer is disabled.

26.6.5.2.3 Slot Status Description

The slot status is a read-only structure for the application and a write-only structure for the controller. The meaning and content of the slot status in the message buffer header field depends on the message buffer type.

Receive Message Buffer and Receive FIFO Slot Status Description

This section describes the slot status structure for the individual receive message buffers and receive FIFOs. The content of the slot status structure for receive message buffers depends on the message buffer type and on the channel assignment for individual receive message buffers as given by Table 26-87.

Table 26-87. Receive Message Buffer Slot Status Content

Receive Message Buffer Type	Slot Status Content
Individual Receive Message Buffer assigned to both channels MBCCSR*n*[CHA] = 1 and MBCCSR*n*[CHB] = 1	see Figure 26-112
Individual Receive Message Buffer assigned to channel A MBCCSR*n*[CHA] = 1 and MBCCSR*n*[CHB] = 0	see Figure 26-113
Individual Receive Message Buffer assigned to channel B MBCCSR*n*[CHA] = 0 and MBCCSR*n*[CHB] = 1	see Figure 26-114
Receive FIFO Channel A Message Buffer	see Figure 26-113
Receive FIFO Channel B Message Buffer	see Figure 26-114

The meaning of the bits in the slot status structure is explained in Table 26-88.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	VFB	SYB	NFB	SUB	SEB	CEB	BVB	CH	VFA	SYA	NFA	SUA	SEA	CEA	BVA	0
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Figure 26-112. Receive Message Buffer Slot Status Structure (ChAB)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	VFA	SYA	NFA	SUA	SEA	CEA	BVA	0
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Figure 26-113. Receive Message Buffer Slot Status Structure (ChA)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	VFB	SYB	NFB	SUB	SEB	CEB	BVB	1	0	0	0	0	0	0	0	0
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Figure 26-114. Receive Message Buffer Slot Status Structure (ChB)

Table 26-88. Receive Message Buffer Slot Status Field Descriptions

Field	Description
Common Message Buffer Status Bits	
VFB	**Valid Frame on Channel B** — protocol related variable: *vSS!ValidFrame* channel B. 0 *vSS!ValidFrame* = 0. 1 *vSS!ValidFrame* = 1.
SYB	**Sync Frame Indicator Channel B** — protocol related variable: *vRF!Header!SyFIndicator* channel B. 0 *vRF!Header!SyFIndicator* = 0. 1 *vRF!Header!SyFIndicator* = 1.
NFB	**Null Frame Indicator Channel B** — protocol related variable: *vRF!Header!NFIndicator* channel B. 0 *vRF!Header!NFIndicator* = 0. 1 *vRF!Header!NFIndicator* = 1.
SUB	**Startup Frame Indicator Channel B** — protocol related variable: *vRF!Header!SuFIndicator* channel B. 0 *vRF!Header!SuFIndicator* = 0. 1 *vRF!Header!SuFIndicator* = 1.
SEB	**Syntax Error on Channel B** — protocol related variable: *vSS!SyntaxError* channel B. 0 *vSS!SyntaxError* = 0. 1 *vSS!SyntaxError* = 1.
CEB	**Content Error on Channel B** — protocol related variable: *vSS!ContentError* channel B. 0 *vSS!ContentError* = 0. 1 *vSS!ContentError* = 1.
BVB	**Boundary Violation on Channel B** — protocol related variable: *vSS!BViolation* channel B. 0 *vSS!BViolation* = 0. 1 *vSS!BViolation* = 1.
CH	**Channel first valid received** — This status bit applies only to receive message buffers assigned to the static segment and to both channels. It indicates the channel that has received the *first valid* frame in the slot. This flag is set to 0 if no valid frame was received at all in the subscribed slot. 0 first valid frame received on channel A, or no valid frame received at all. 0 first valid frame received on channel B.
VFA	**Valid Frame on Channel A** — protocol related variable: *vSS!ValidFrame* channel A. 0 *vSS!ValidFrame* = 0. 1 *vSS!ValidFrame* = 1.

Table 26-88. Receive Message Buffer Slot Status Field Descriptions (continued)

Field	Description
SYA	**Sync Frame Indicator Channel A** — protocol related variable: *vRF!Header!SyFIndicator* channel A. 0 *vRF!Header!SyFIndicator* = 0. 1 *vRF!Header!SyFIndicator* = 1.
NFA	**Null Frame Indicator Channel A** — protocol related variable: *vRF!Header!NFIndicator* channel A. 0 *vRF!Header!NFIndicator* = 0. 1 *vRF!Header!NFIndicator* = 1.
SUA	**Startup Frame Indicator Channel A** — protocol related variable: *vRF!Header!SuFIndicator* channel A. 0 *vRF!Header!SuFIndicator* = 0. 1 *vRF!Header!SuFIndicator* = 1.
SEA	**Syntax Error on Channel A** — protocol related variable: *vSS!SyntaxError* channel A. 0 *vSS!SyntaxError* = 0. 1 *vSS!SyntaxError* = 1.
CEA	**Content Error on Channel A** — protocol related variable: *vSS!ContentError* channel A. 0 *vSS!ContentError* = 0. 1 *vSS!ContentError* = 1.
BVA	**Boundary Violation on Channel A** — protocol related variable: *vSS!BViolation* channel A. 0 *vSS!BViolation* = 0. 1 *vSS!BViolation* = 1.

Transmit Message Buffer Slot Status Description

This section describes the slot status structure for transmit message buffers. Only the TCA and TCB status bits are directly related to the transmission process. All other status bits in this structure are related to a receive process that may have occurred. The content of the slot status structure for transmit message buffers depends on the channel assignment as given by Table 26-89.

Table 26-89. Transmit Message Buffer Slot Status Content

Transmit Message Buffer Type	Slot Status Content
Individual Transmit Message Buffer assigned to both channels MBCCSR*n*[CHA] = 1 and MBCCSR*n*[CHB] = 1	see Figure 26-115
Individual Transmit Message Buffer assigned to channel A MBCCSR*n*[CHA] = 1 and MBCCSR*n*[CHB] = 0	see Figure 26-116
Individual Transmit Message Buffer assigned to channel B MBCCSR*n*[CHA] = 0 and MBCCSR*n*[CHB] = 1	see Figure 26-117

The meaning of the bits in the slot status structure is described in Table 26-88.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	VFB	SYB	NFB	SUB	SEB	CEB	BVB	TCB	VFA	SYA	NFA	SUA	SEA	CEA	BVA	TCA
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Figure 26-115. Transmit Message Buffer Slot Status Structure (ChAB)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	VFA	SYA	NFA	SUA	SEA	CEA	BVA	TCA
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Figure 26-116. Transmit Message Buffer Slot Status Structure (ChA)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	VFB	SYB	NFB	SUB	SEB	CEB	BVB	TCB	0	0	0	0	0	0	0	0
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Figure 26-117. Transmit Message Buffer Slot Status Structure (ChB)

Table 26-90. Transmit Message Buffer Slot Status Structure Field Descriptions

Field	Description
VFB	**Valid Frame on Channel B** — protocol related variable: *vSS!ValidFrame* channel B. 0 *vSS!ValidFrame* = 0. 1 *vSS!ValidFrame* = 1.
SYB	**Sync Frame Indicator Channel B** — protocol related variable: *vRF!Header!SyFIndicator* channel B. 0 *vRF!Header!SyFIndicator* = 0. 1 *vRF!Header!SyFIndicator* = 1.
NFB	**Null Frame Indicator Channel B** — protocol related variable: *vRF!Header!NFIndicator* channel B. 0 *vRF!Header!NFIndicator* = 0. 1 *vRF!Header!NFIndicator* = 1.
SUB	**Startup Frame Indicator Channel B** — protocol related variable: *vRF!Header!SuFIndicator* channel B. 0 *vRF!Header!SuFIndicator* = 0. 1 *vRF!Header!SuFIndicator* = 1.
SEB	**Syntax Error on Channel B** — protocol related variable: *vSS!SyntaxError* channel B. 0 *vSS!SyntaxError* = 0. 1 *vSS!SyntaxError* = 1.
CEB	**Content Error on Channel B** — protocol related variable: *vSS!ContentError* channel B. 0 *vSS!ContentError* = 0. 1 *vSS!ContentError* = 1.
BVB	**Boundary Violation on Channel B** — protocol related variable: *vSS!BViolation* channel B. 0 *vSS!BViolation* = 0. 1 *vSS!BViolation* = 1.
TCB	**Transmission Conflict on Channel B** — protocol related variable: *vSS!TxConflict* channel B. 0 *vSS!TxConflict* = 0. 1 *vSS!TxConflict* = 1.
VFA	**Valid Frame on Channel A** — protocol related variable: *vSS!ValidFrame* channel A. 0 *vSS!ValidFrame* = 0. 1 *vSS!ValidFrame* = 1.
SYA	**Sync Frame Indicator Channel A** — protocol related variable: *vRF!Header!SyFIndicator* channel A. 0 *vRF!Header!SyFIndicator* = 0. 1 *vRF!Header!SyFIndicator* = 1.
NFA	**Null Frame Indicator Channel A** — protocol related variable: *vRF!Header!NFIndicator* channel A. 0 *vRF!Header!NFIndicator* = 0. 1 *vRF!Header!NFIndicator* = 1.
SUA	**Startup Frame Indicator Channel A** — protocol related variable: *vRF!Header!SuFIndicator* channel A. 0 *vRF!Header!SuFIndicator* = 0. 1 *vRF!Header!SuFIndicator* = 1.

Table 26-90. Transmit Message Buffer Slot Status Structure Field Descriptions (continued)

Field	Description
SEA	**Syntax Error on Channel A** — protocol related variable: *vSS!SyntaxError* channel A. 0 *vSS!SyntaxError* = 0. 1 *vSS!SyntaxError* = 1.
CEA	**Content Error on Channel A** — protocol related variable: *vSS!ContentError* channel A. 0 *vSS!ContentError* = 0. 1 *vSS!ContentError* = 1.
BVA	**Boundary Violation on Channel A** — protocol related variable: *vSS!BViolation* channel A. 0 *vSS!BViolation* = 0. 1 *vSS!BViolation* = 1.
TCA	**Transmission Conflict on Channel A** — protocol related variable: *vSS!TxConflict* channel A. 0 *vSS!TxConflict* = 0. 1 *vSS!TxConflict* = 1.

26.6.5.3 Message Buffer Data Field Description

The message buffer data field is used to store the frame payload data, or a part of it, of the frame to be transmitted to or received from the FlexRay bus. The minimum required length of this field depends on the message buffer type that the physical message buffer is assigned to and is given in Table 26-91. The structure of the message buffer data field is given in Figure 26-118.

Table 26-91. Message Buffer Data Field Minimum Length

physical message buffer assigned to	minimum length defined by
Individual Message Buffer in Segment 1	MBDSR[MBSEG1DS]
Receive Shadow Buffer in Segment 1	MBDSR[MBSEG1DS]
Individual Message Buffer in Segment 2	MBDSR[MBSEG2DS]
Receive Shadow Buffer in Segment 2	MBDSR[MBSEG2DS]
Receive FIFO for channel A	RFDSR[ENTRY_SIZE] (RFSR[SEL] = 0)
Receive FIFO for channel B	RFDSR[ENTRY_SIZE] (RFSR[SEL] = 1)

NOTE

The controller will not access any locations outside the message buffer data field boundaries given by Table 26-91.

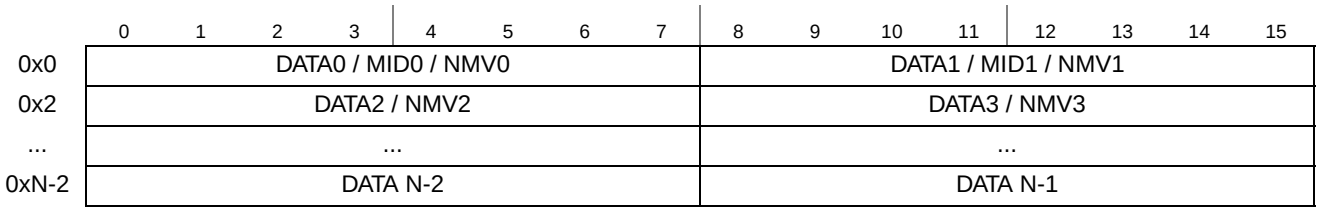


Figure 26-118. Message Buffer Data Field Structure

The message buffer data field is located in the FlexRay memory; thus, the controller has no means to control application write access to the field. To ensure data consistency, the application must follow a write and read access scheme.

26.6.5.3.1 Message Buffer Data Field Read Access

For transmit message buffers, the controller will not modify the content of the Message Buffer Data Field. Thus the application can read back the data at any time without any impact on data consistency.

For receive message buffers the application must lock the related receive message buffer and retrieve the message buffer header index from the Message Buffer Index Registers (MBIDXRn). While the message buffer is locked, the controller will not update the Message Buffer Data Field.

For receive FIFOs, the application can read the message buffer indicated by the Receive FIFO A Read Index Register (RFARIR) or the Receive FIFO B Read Index Register (RFBRIR) when the related fill levels in the Receive FIFO Fill Level and POP Count Register (RFFLPCR) indicate an non-empty FIFO.

26.6.5.3.2 Message Buffer Data Field Write Access

For receive message buffers, receive shadow buffers, and receive FIFOs, the application must not write to the message buffer data field.

For transmit message buffers, the application must follow the write access restrictions given in Table 26-92.

Table 26-92. Frame Data Write Access Constraints

Field	single buffered	double buffered	
		commit side	transmit side
DATA, MID, NMV	*POC:config* or MB_DIS or MB_LCK	*POC:config* or MB_DIS or MB_LCK	*POC:config* or MB_DIS

Table 26-93. Frame Data Field Descriptions

Field	Description
DATA 0, DATA 1, ... DATA N-1	**Message Data** — Provides the message data received or to be transmitted. For receive message buffer and receive FIFOs, this field provides the message data received for this message buffer. For transmit message buffers, the field provides the message data to be transmitted.
MID 0, MID 1	Message Identifier — If the payload preamble bit PPI is set in the message buffer frame header, the MID field holds the message ID of a dynamic frame located in the message buffer. The receive FIFO filter uses the received message ID for message ID filtering.
NMV 0, NMV 1, ... NMV 11	Network Management Vector — If the payload preamble bit PPI is set in the message buffer frame header, the network management vector field holds the network management vector of a static frame located in the message buffer. **Note:** The MID and NMV bytes replace the corresponding DATA bytes.

26.6.6 Individual Message Buffer Functional Description

The controller provides three basic types of individual message buffers:

1. Single Transmit Message Buffers
2. Double Transmit Message Buffers
3. Receive Message Buffers

Before an individual message buffer can be used, it must be configured by the application. After the initial configuration, the message buffer can be reconfigured later. The set of the configuration data for individual message buffers is given in Section 26.6.3.4.1, Individual Message Buffer Configuration Data.

26.6.6.1 Individual Message Buffer Configuration

The individual message buffer configuration consists of two steps. The first step is the allocation of the required amount of memory for the FlexRay memory. The second step is the programming of the message buffer configuration registers, which is described in this section.

26.6.6.1.1 Common Configuration Data

One part of the message buffer configuration data is common to all individual message buffers and the receive shadow buffers. These data can only be set when the protocol is in the *POC:config* state.

The application configures the number of utilized individual message buffers by writing the message buffer number of the last utilized message buffer into the LAST_MB_UTIL field in the Message Buffer Segment Size and Utilization Register (MBSSUTR).

The application configures the size of the two segments of individual message buffers by writing the message buffer number of the last message buffer in the first segment into the LAST_MB_SEG1 field in the Message Buffer Segment Size and Utilization Register (MBSSUTR)

The application configures the length of the message buffer data fields for both of the message buffer segments by writing to the MBSEG2DS and MBSEG1DS fields in the Message Buffer Data Size Register (MBDSR).

Depending on the current receive functionality of the controller, the application must configure the receive shadow buffers. For each segment and for each channel with at least one individual receive message buffer assigned, the application must configure the related receive shadow buffer using the Receive Shadow Buffer Index Register (RSBIR).

26.6.6.1.2 Specific Configuration Data

The second part of the message buffer configuration data is specific for each message buffer.

These data can be changed only when either

- the protocol is in the *POC:config* state or
- the message buffer is disabled (MBCCSR*n*[EDS] = 0)

The individual message buffer type is defined by the MTD and MBT bits in the Message Buffer Configuration, Control, Status Registers (MBCCSRn) as given in Table 26-94.

Table 26-94. Individual Message Buffer Types

MBCCSR*n*[MTD]	MBCCSR*n*[MBT]	Individual Message Buffer Description
0	0	Receive Message Buffer
0	1	Reserved
1	0	Single Transmit Message Buffer
1	1	Double Transmit Message Buffer

The message buffer specific configuration data are

1. MCM, MBT, MTD bits in Message Buffer Configuration, Control, Status Registers (MBCCSRn)
2. all fields and bits in Message Buffer Cycle Counter Filter Registers (MBCCFRn)
3. all fields and bits in Message Buffer Frame ID Registers (MBFIDRn)
4. all fields and bits in Message Buffer Index Registers (MBIDXRn)

The meaning of the specific configuration data depends on the message buffer type, as given in the detailed message buffer type descriptions Section 26.6.6.2, Single Transmit Message Buffers, Section 26.6.6.3, Receive Message Buffers, and Section 26.6.6.4, Double Transmit Message Buffer.

26.6.6.2 Single Transmit Message Buffers

The section provides a detailed description of the functionality of single buffered transmit message buffers.

A single transmit message buffer is used by the application to provide message data to the controller that will be transmitted over the FlexRay Bus. The controller uses the transmit message buffers to provide information about the transmission process and status information about the slot in which message was transmitted.

The individual message buffer with message buffer number *n* is configured to be a single transmit message buffer by the following settings:

- MBCCSR*n*[MBT] = 0 (single buffered message buffer)
- MBCCSR*n*[MTD] = 1 (transmit message buffer)

26.6.6.2.1 Access Regions

To certain message buffer fields, both the application and the controller have access. To ensure data consistency, a message buffer locking scheme is implemented, which is used to control the access to the data, control, and status bits of a message buffer. The access regions for single transmit message buffers are depicted in Figure 26-119. A description of the regions is given in Table 26-95. If an region is active as indicated in Table 26-96, the access scheme given for that region applies to the message buffer.

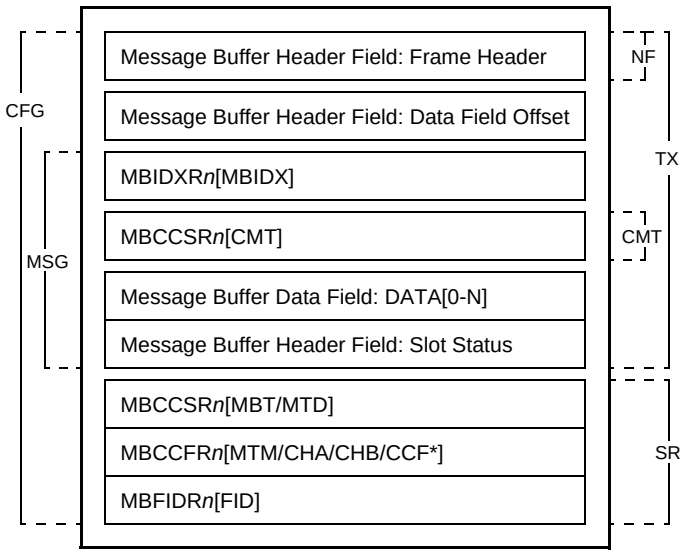


Figure 26-119. Single Transmit Message Buffer Access Regions

Table 26-95. Single Transmit Message Buffer Access Regions Description

Region	Access from		Region used for
	Application	Module	
CFG	read/write	-	Message Buffer Configuration
MSG	read/write	-	Message Data and Slot Status Access
NF	-	read-only	Message Header Access for Null Frame Transmission
TX	-	read/write	Message Transmission and Slot Status Update
CM	-	read-only	Message Buffer Validation
SR	-	read-only	Message Buffer Search

The trigger bits MBCCSR*n*[EDT] and MBCCSR*n*[LCKT], and the interrupt enable bit MBCCSR*n*[MBIE] are not under access control and can be accessed from the application at any time. The status bits MBCCSR*n*[EDS] and MBCCSR*n*[LCKS] are not under access control and can be accessed from the controller at any time.

The interrupt flag MBCCSR*n*[MBIF] is not under access control and can be accessed from the application and the controller at any time. controller clear access has higher priority.

The controller restricts its access to the regions depending on the current state of the message buffer. The application must adhere to these restrictions in order to ensure data consistency. The transmit message buffer states are given in Figure 26-120. A description of the states is given in Table 26-96, which also provides the access scheme for the access regions.

The status bits MBCCSR*n*[EDS] and MBCCSR*n*[LCKS] provide the application with the required message buffer status information. The internal status information is not visible to the application.

26.6.6.2.2 Message Buffer States

This section describes the transmit message buffer states and provides a state diagram.

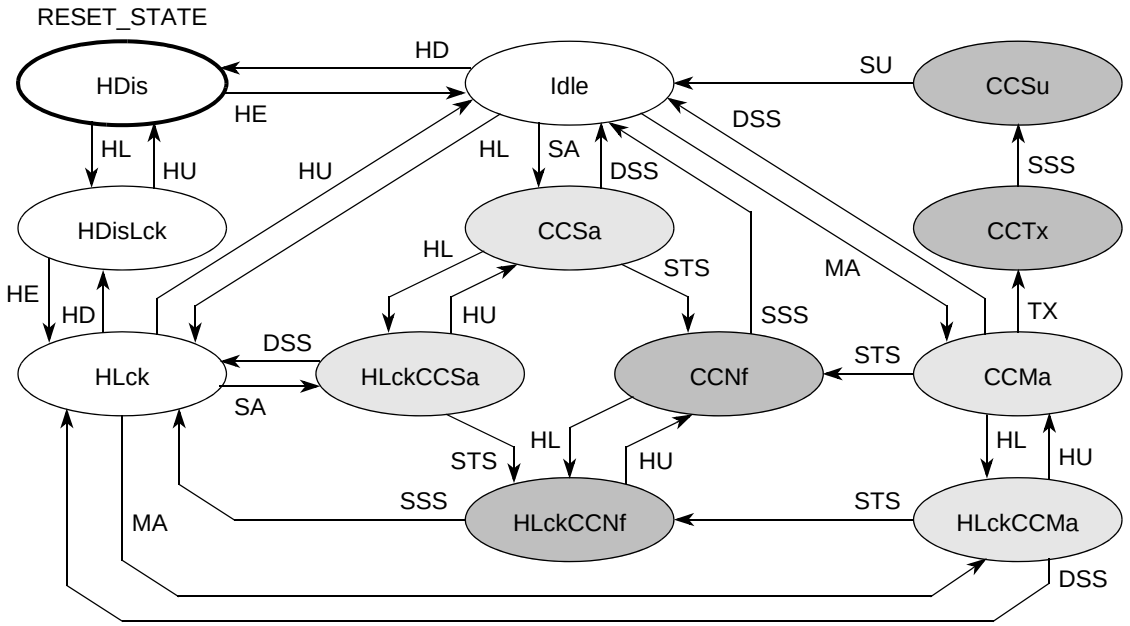


Figure 26-120. Single Transmit Message Buffer States

Table 26-96. Single Transmit Message Buffer State Description

State	MBCCSR*n*		Access Region		Description
	EDS	LCKS	Appl.	Module	
Idle	1	0	–	CM, SR	**Idle** - Message Buffer is idle. Included in message buffer search.
HDis	0	0	CFG	–	**Disabled** - Message Buffer under configuration. Excluded from message buffer search.
HDisLck	0	1	CFG	–	**Disabled and Locked** - Message Buffer under configuration. Excluded from message buffer search.
HLck	1	1	MSG	SR	Locked - Applications access to data, control, and status. Included in message buffer search.
CCSa	1	0	–	–	Slot Assigned - Message buffer assigned to next static slot. Ready for Null Frame transmission.
HLckCCSa	1	1	MSG	–	Locked and Slot Assigned - Applications access to data, control, and status.Message buffer assigned to next static slot
CCNf	1	0	–	NF	Null Frame Transmission Header is used for null frame transmission.
HLckCCNf	1	1	MSG	NF	Locked and Null Frame Transmission - Applications access to data, control, and status. Header is used for null frame transmission.
CCMa	1	0	–	CM	Message Available - Message buffer is assigned to next slot and cycle counter filter matches.
HLckCCMa	1	1	MSG	–	Locked and Message Available - Applications access to data, control, and status. Message buffer is assigned to next slot and cycle counter filter matches.
CCTx	1	0	–	TX	Message Transmission - Message buffer data transmit. Payload data from buffer transmitted

Table 26-96. Single Transmit Message Buffer State Description (continued)

State	MBCCSR*n*		Access Region		Description
	EDS	LCKS	Appl.	Module	
CCSu	1	0	–	TX	Status Update - Message buffer status update. Update of status flags, the slot status field, and the header index.

26.6.6.2.3 Message Buffer Transitions

Application Transitions

The application transitions can be triggered by the application using the commands described in Table 26-97. The application issues the commands by writing to the Message Buffer Configuration, Control, Status Registers (MBCCSRn). Only one command can be issued with one write access. Each command is executed immediately. If the command is ignored, it must be issued again.

Message Buffer Enable and Disable

The enable and disable commands issued by writing 1 to the trigger bit MBCCSR*n*[EDT]. The transition that will be triggered by each of these command depends on the current value of the status bit MBCCSR*n*[EDS]. If the command triggers the disable transition HD and the message buffer is in one of the states CCSa, HLckCCSa, CCMa, HLckCCMa, CCNf, HLckCCNf, or CCTx, the disable transition has no effect (command is ignored) and the message buffer state is not changed. No notification is given to the application.

Message Buffer Lock and Unlock

The lock and unlock commands issued by writing 1 to the trigger bit MBCCSR*n*[LCKT]. The transition that will be triggered by each of these commands depends on the current value of the status bit MBCCSR*n*[LCKS]. If the command triggers the lock transition HL and the message buffer is in the state CCTx, the lock transition has no effect (command is ignored) and message buffer state is not changed. In this case, the message buffer lock error flag LCK_EF in the CHI Error Flag Register (CHIERFR) is set.

Table 26-97. Single Transmit Message Buffer Application Transitions

Transition	Command	Condition	Description
HE	MBCCSR*n*[EDT] = 1	MBCCSR*n*[EDS] = 0	Application triggers message buffer enable.
HD		MBCCSR*n*[EDS] = 1	Application triggers message buffer disable.
HL	MBCCSR*n*[LCKT] = 1	MBCCSR*n*[LCKS] = 0	Application triggers message buffer lock.
HU		MBCCSR*n*[LCKS] = 1	Application triggers message buffer unlock.

Module Transitions

The module transitions that can be triggered by the controller are described in Table 26-98. Each transition will be triggered for certain message buffers when the related condition is fulfilled.

Table 26-98. Single Transmit Message Buffer Module Transitions

Transition	Condition	Description
SA	slot match and static slot	Slot Assigned - Message buffer is assigned to next static slot.

Table 26-98. Single Transmit Message Buffer Module Transitions (continued)

Transition	Condition	Description
MA	slot match and CycleCounter match	Message Available - Message buffer is assigned to next slot and cycle counter filter matches.
TX	slot start and MBCCSR*n*[CMT] = 1	Transmission Slot Start - Slot Start and commit bit CMT is set. In case of a dynamic slot, pLatestTx is not exceeded.
SU	status updated	Status Updated - Slot Status field and message buffer status flags updated. Interrupt flag set.
STS	static slot start	Static Slot Start - Start of static slot.
DSS	dynamic slot start or symbol window start or NIT start	Dynamic Slot or Segment Start. - Start of dynamic slot or symbol window or NIT.
SSS	slot start or symbol window start or NIT start	Slot or Segment Start - Start of static slot or dynamic slot or symbol window or NIT.

Transition Priorities

The application can trigger only one transition at a time. There is no need to specify priorities among them.

As shown in the first part of Table 26-99, the module transitions have a higher priority than the application transitions. For all states except the CCMa state, both a lock/unlock transition HL/HD and a module transition can be executed at the same time. The result state is reached by first applying the application transition and subsequently the module transition to the intermediately reached state. For example, if the message buffer is in the HLck state and the application unlocks the message buffer by the HU transition and the module triggers the slot assigned transition SA, the intermediate state is Idle and the resulting state is CCSa.

The priorities among the module transitions is given in the second part of Table 26-99.

Table 26-99. Single Transmit Message Buffer Transition Priorities

State	Priorities	Description
module vs. application		
Idle, HLck	SA > HD MA > HD	Slot Assigned > Message Buffer Disable Message Available > Message Buffer Disable
CCMa	TX > HL	Transmission Start > Message Buffer Lock
module internal		
Idle, HLck	MA > SA	Message Available > Slot Assigned
CCMa	TX > STS TX > DSS	Transmission Slot Start > Static Slot Start Transmission Slot Start > Dynamic Slot Start

26.6.6.2.4 Transmit Message Setup

To transmit a message over the FlexRay bus, the application writes the message data into the message buffer data field and sets the commit bit CMT in the Message Buffer Configuration, Control, Status Registers (MBCCSRn). The physical access to the message buffer data field is described in Section 26.6.3.1, Individual Message Buffers.

As indicated by Table 26-96, the application shall write to the message buffer data field and change the commit bit CMT only if the transmit message buffer is in one of the states HDis, HDisLck, HLck, HLckCCSa, HLckCCMa, or HLckCCMa. The application can change the state of a message buffer if it issues the appropriate commands shown in Table 26-97. The state change is indicated through the MBCCSR*n*[EDS] and MBCCSR*n*[LCKS] status bits.

If the transmit message buffer enters one of the states HDis, HDisLck, HLck, HLckCCSa, HLckCCMa, or HLckCCMa the MBCCSR*n*[DVAL] flag is negated.

26.6.6.2.5 Message Transmission

As a result of the message buffer search described in Section 26.6.7, Individual Message Buffer Search, the controller triggers the message available transition MA for as many as two transmit message buffers. This changes the message buffer state from Idle to CCMa and the message buffers can be used for message transmission in the next slot.

The controller transmits a message from a message buffer if both of the following two conditions are fulfilled at the start of the transmission slot:

1. the message buffer is in the message available state CCMa
2. the message data are still valid (MBCCSR*n*[CMT] = 1)

In this case, the controller triggers the TX transition and changes the message buffer state to CCTx. A transmit message buffer timing and state change diagram for message transmission is given in Figure 26-121. In this example, the message buffer with message buffer number *n* is Idle at the start of the search slot, matches the slot and cycle number of the next slot, and message buffer data are valid (MBCCSR*n*[CMT] = 1).

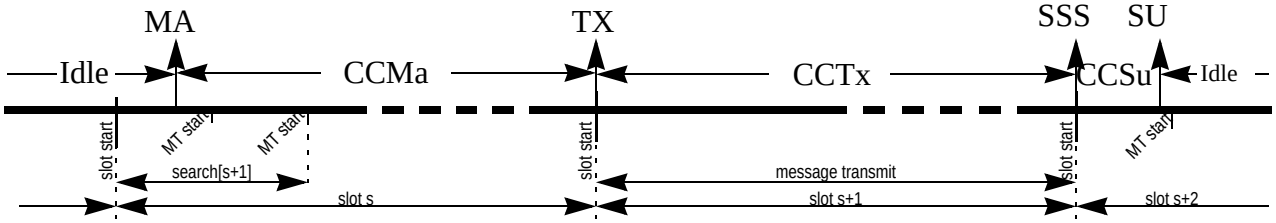


Figure 26-121. Message Transmission Timing

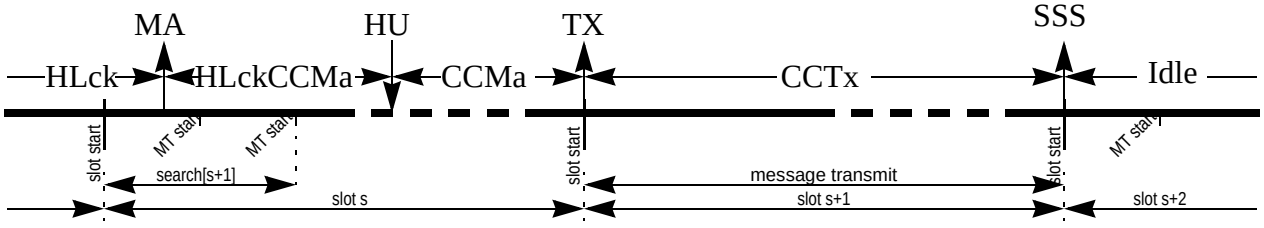


Figure 26-122. Message Transmission from HLck state with unlock

The amount of message data read from the FlexRay memory and transferred to the FlexRay bus is determined by the following three items

1. the message buffer segment that the message buffer is assigned to, as defined by the Message Buffer Segment Size and Utilization Register (MBSSUTR).
2. the message buffer data field size, as defined by the related field of the Message Buffer Data Size Register (MBDSR)
3. the value of the PLDLEN field in the message buffer header field, as described in Section 26.6.5.2.1, Frame Header Description.

If a message buffer is assigned to message buffer segment 1, and PLDLEN > MBSEG1DS, then 2 * MBSEG1DS bytes will be read from the message buffer data field and zero padding is used for the remaining bytes for the FlexRay bus transfer. If PLDLEN ≤ MBSEG1DS, the controller reads and transfers 2*PLDLEN bytes. The same holds for segment 2 and MBSEG2DS.

26.6.6.2.6 Null Frame Transmission

A static slot with slot number S is assigned to the controller for channel A, if at least one transmit message buffer is configured with the MBFIDR*n*[FID] set to S and MBCCFR*n*[CHA] set to 1. A Null Frame is transmitted in the static slot S on channel A, if this slot is assigned to the controller for channel A, and all transmit message buffers with MBFIDR*n*[FID] = S and MBCCFR*n*[CHA] = 1 are either not committed (MBCCSR*n*[CMT] = 0), or locked by the application (MBCCSR*n*[LCKS] = 1), or the cycle counter filter is enabled and does not match.

Additionally, the application can clear the commit bit of a message buffer that is in the CCMa state, which is called *uncommit* or *transmit abort*. This message buffer will be used for null frame transmission.

As a result of the message buffer search described in Section 26.6.7, Individual Message Buffer Search, the controller triggers the slot assigned transition SA for as many as two transmit message buffers if at least one of the conditions mentioned above is fulfilled for these message buffers. The transition SA changes the message buffer states from either Idle to CCSa or from HLck to HLckCCSa. In each case, these message buffers will be used for null frame transmission in the next slot. A message buffer timing and state change diagram for null frame transmission from Idle state is given in Figure 26-123.

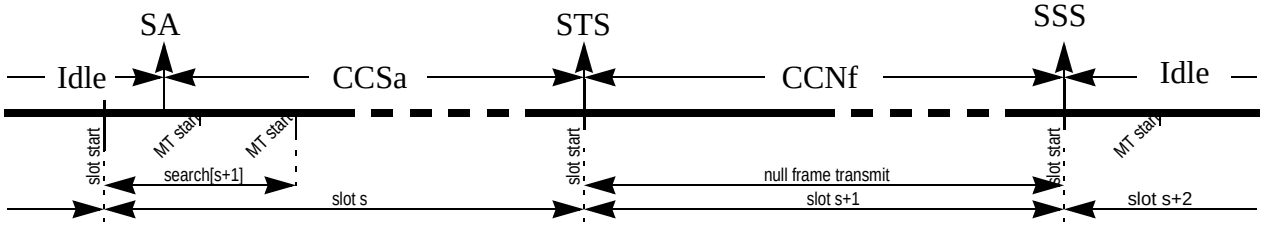


Figure 26-123. Null Frame Transmission from Idle state

A message buffer timing and state change diagram for null frame transmission from HLck state is given in Figure 26-124.

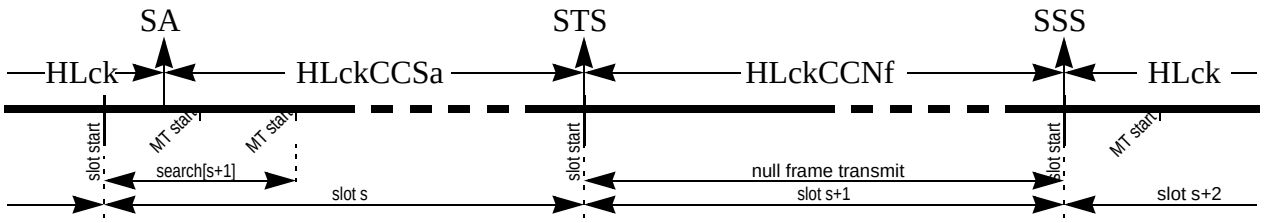


Figure 26-124. Null Frame Transmission from HLck state

If a transmit message buffer is in the CCSa or HLckCCSa state at the start of the transmission slot, a null frame is transmitted in any case, even if the message buffer is unlocked or committed before the transmission slot starts. A transmit message buffer timing and state change diagram for null frame transmission for this case is given in Figure 26-125.

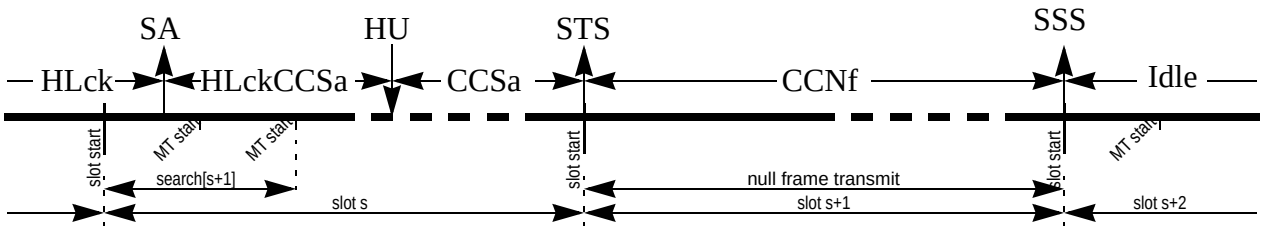


Figure 26-125. Null Frame Transmission from HLck state with unlock

Since the null frame transmission will not use the message buffer data, the application can lock/unlock the message buffer during null frame transmission. A transmit message buffer timing and state change diagram for null frame transmission for this case is given in Figure 26-126.

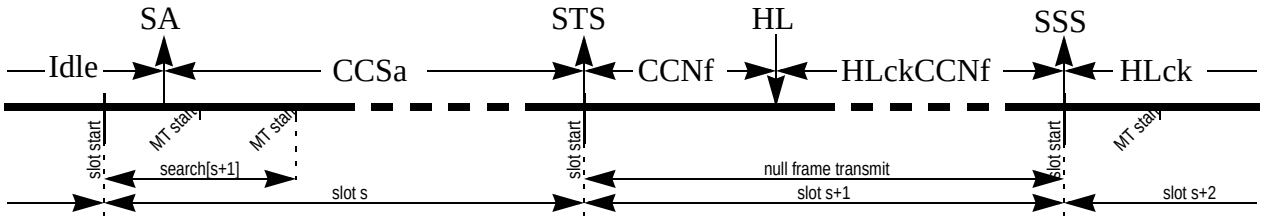


Figure 26-126. Null Frame Transmission from Idle state with locking

26.6.6.2.7 Message Buffer Status Update

After the end of each slot, the PE generates the slot status vector. Depending on the this status, the transmitted frame type, and the amount of transmitted data, the message buffer status is updated.

Message Buffer Status Update after Complete Message Transmission

The term complete message transmission refers to the fact that all payload data stored in the message buffer were send to FlexRay bus. In this case, the controller updates the slot status field of the message

buffer and triggers the status updated transition SU. With the SU transition, the controller sets the message buffer interrupt flag MBCCSR*n*[MBIF] to indicate the successful message transmission.

Depending on the transmission mode flag MBCCFR*n*[MTM], the controller changes the commit flag MBCCSR*n*[CMT] and the valid flag MBCCSR*n*[DVAL]. If the MBCCFR*n*[MTM] flag is negated, the message buffer is in the *event transmission mode*. In this case, each committed message is transmitted only once. The commit flag MBCCSR*n*[CMT] is cleared with the SU transition. If the MBCCFR*n*[MTM] flag is asserted, the message buffer is in the *state transmission mode*. In this case, each committed message is transmitted as long as the application provides new data or locks the message buffers. The controller will not clear the MBCCSR*n*[CMT] flag at the end of transmission and will set the valid flag MBCCSR*n*[DVAL] to indicate that the message will be transmitted again.

Message Buffer Status Update after Incomplete Message Transmission

The term incomplete message transmission refers to the fact that not all payload data that should be transmitted were send to FlexRay bus. This may be caused by the following regular conditions in the dynamic segment:

1. The transmission slot starts in a minislot with a minislot number greater than *pLatestTx*.
2. The transmission slot did not exist in the dynamic segment at all.

Additionally, an incomplete message transmission can be caused by internal communication errors. If those error occur, the Protocol Engine Communication Failure Interrupt Flag PECF_IF is set in the Protocol Interrupt Flag Register 1 (PIFR1).

In any of these two cases, the status of the message buffer is not changed at all with the SU transition. The slot status field is not updated, the status and control flags are not changed, and the interrupt flag is not set.

Message Buffer Status Update after Null Frame Transmission

After the transmission of a null frame, the status of the message buffer that was used for the null frame transmission is not changed at all. The slot status field is not updated, the status and control flags are not changed, and the interrupt flag is not set.

26.6.6.3 Receive Message Buffers

The section provides a detailed description of the functionality of the receive message buffers.

A receive message buffer is used to receive a message from the FlexRay Bus based on individual filter criteria. The controller uses the receive message buffer to provide the following data to the application

1. message data received
2. information about the reception process
3. status information about the slot in which the message was received

A individual message buffer with message buffer number *n* is configured as a receive message buffer by the following configuration settings

- MBCCSR*n*[MBT] = 0 (single buffered message buffer)
- MBCCSR*n*[MTD] = 0 (receive message buffer)

To certain message buffer fields, both the application and the controller have access. To ensure data consistency, a message buffer locking scheme is implemented that is used to control the access to the data, control, and status bits of a message buffer. The access regions for receive message buffers are depicted in Figure 26-127. A description of the regions is given in Table 26-100. If an region is active as indicated in Table 26-101, the access scheme given for that region applies to the message buffer.

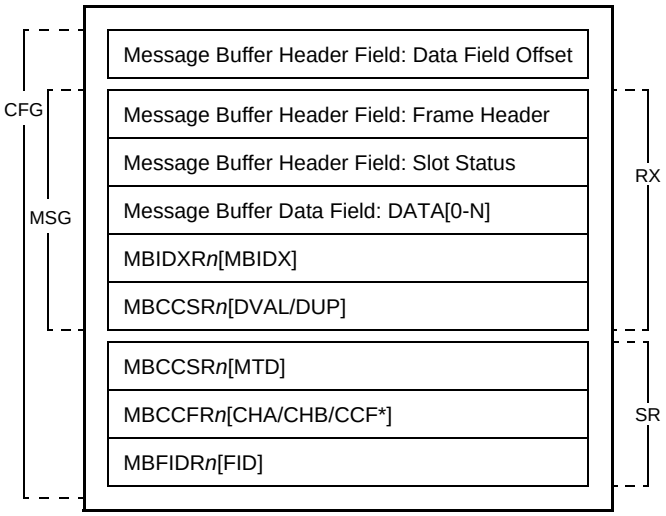


Figure 26-127. Receive Message Buffer Access Regions

Table 26-100. Receive Message Buffer Access Region Description

Region	Access from		Region used for
	Application	Module	
CFG	read/write	—	Message Buffer Configuration, Message Data and Status Access
MSG	read/write	—	Message Data, Header, and Status Access
RX	—	write-only	Message Reception and Status Update
SR	—	read-only	Message Buffer Search Data

The trigger bits MBCCSR*n*[EDT] and MBCCSR*n*[LCKT] and the interrupt enable bit MBCCSR*n*[MBIE] are not under access control and can be accessed from the application at any time. The status bits MBCCSR*n*[EDS] and MBCCSR*n*[LCKS] are not under access control and can be accessed from the controller at any time.

The interrupt flag MBCCSR*n*[MBIF] is not under access control and can be accessed from the application and the controller at any time. controller set access has higher priority.

The controller restricts its access to the regions depending on the current state of the message buffer. The application must adhere to these restrictions in order to ensure data consistency. The receive message buffer states are given in Figure 26-128. A description of the message buffer states is given in Table 26-96, which also provides the access scheme for the access regions.

The status bits MBCCSR*n*[EDS] and MBCCSR*n*[LCKS] provide the application with the required status information. The internal status information is not visible to the application.

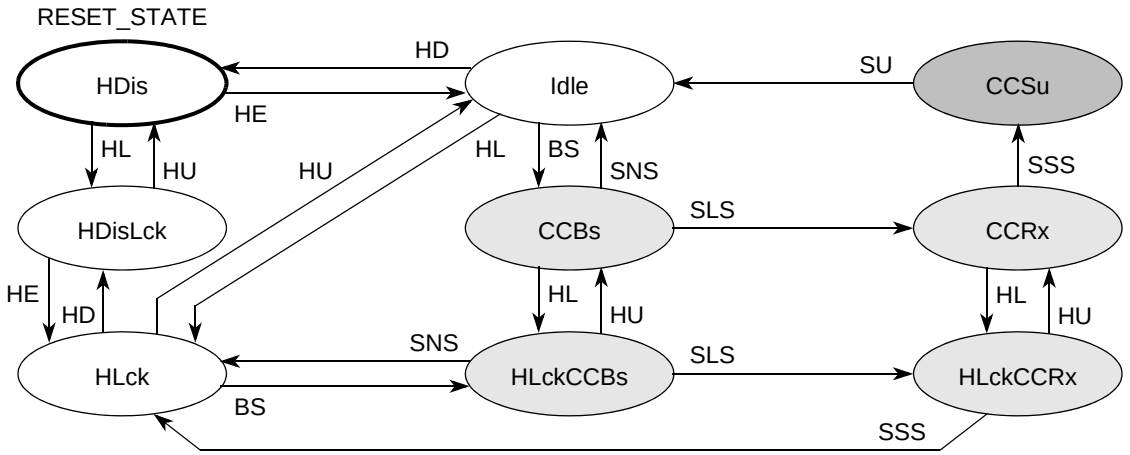


Figure 26-128. Receive Message Buffer States

Table 26-101. Receive Message Buffer States and Access

State	MBCCSR*n*		Access from		Description
	EDS	LCKS	Appl.	Module	
Idle	1	0	–	SR	**Idle** - Message Buffer is idle. Included in message buffer search.
HDis	0	0	CFG	–	**Disabled** - Message Buffer under configuration. Excluded from message buffer search.
HDisLck	0	1	CFG	–	**Disabled and Locked** - Message Buffer under configuration. Excluded from message buffer search.
HLck	1	1	MSG	–	**Locked** - Applications access to data, control, and status. Included in message buffer search.
CCBs	1	0	–	–	**Buffer Subscribed** - Message buffer subscribed for reception. Filter matches next (slot, cycle, channel) tuple.
HLckCCBs	1	1	MSG	–	**Locked and Buffer Subscribed** - Applications access to data, control, and status. Message buffer subscribed for reception.
CCRx	1	0	–	–	**Message Receive** - Message data received into related shadow buffer.
HLckCCRx	1	1	MSG	–	Locked and Message Receive - Applications access to data, control, and status. Message data received into related shadow buffer.
CCSu	1	0	–	RX	**Status Update** - Message buffer status update. Update of status flags, the slot status field, and the header index.

26.6.6.3.1 Message Buffer Transitions

Application Transitions

The application transitions that can be triggered by the application using the commands described in Table 26-102. The application issues the commands by writing to the Message Buffer Configuration, Control, Status Registers (MBCCSRn). Only one command can be issued with one write access. Each command is executed immediately. If the command is ignored, it must be issued again.

Message Buffer Enable and Disable

The enable and disable commands issued by writing 1 to the trigger bit MBCCSR*n*[EDT]. The transition that will be triggered by each of these command depends on the current value of the status bit MBCCSR*n*[EDS]. If the command triggers the disable transition HD and the message buffer is in one of the states CCBs, HLckCCBs, or CCRx, the disable transition has no effect (command is ignored) and the message buffer state is not changed. No notification is given to the application.

Message Buffer Lock and Unlock

The lock and unlock commands issued by writing 1 to the trigger bit MBCCSR*n*[LCKT]. The transition that will be triggered by each of these commands depends on the current value of the status bit MBCCSR*n*[LCKS]. If the command triggers the lock transition HL while the message buffer is in the state CCRx, the lock transition has no effect (command is ignored) and message buffer state is not changed. In this case, the message buffer lock error flag LCK_EF in the CHI Error Flag Register (CHIERFR) is set.

Table 26-102. Receive Message Buffer Application Transitions

Transition	Host Command	Condition	Description
HE	MBCCSR*n*[EDT] = 1	MBCCSR*n*[EDS] = 0	Application triggers message buffer enable.
HD		MBCCSR*n*[EDS] = 1	Application triggers message buffer disable.
HL	MBCCSR*n*[LCKT] = 1	MBCCSR*n*[LCKS] = 0	Application triggers message buffer lock.
HU		MBCCSR*n*[LCKS] = 1	Application triggers message buffer unlock.

Module Transitions

The module transitions that can be triggered by the controller are described in Table 26-103. Each transition will be triggered for certain message buffers when the related condition is fulfilled.

Table 26-103. Receive Message Buffer Module Transitions

Transition	Condition	Description
BS	slot match and CycleCounter match	Buffer Subscribed - The message buffer filter matches next slot and cycle.
SLS	slot start	Slot Start - Start of either Static Slot or Dynamic Slot.
SNS	symbol window start or NIT start	Symbol Window or NIT Start - Start of either Symbol Window or NIT.
SSS	slot start or symbol window start or NIT start	Slot or Segment Start - Start of either Static Slot, Dynamic Slot, Symbol Window, or NIT.
SU	status updated	Status Updated - Slot Status field, message buffer status flags, header index updated. Interrupt flag set.

Transition Priorities

The application can trigger only one transition at a time. There is no need to specify priorities among them.

As shown in Table 26-104, the module transitions have a higher priority than the application transitions. For all states except the CCRx state, a module transition and the application lock/unlock transition HL/HU and can be executed at the same time. The result state is reached by first applying the module transition and subsequently the application transition to the intermediately reached state. For example, if the message buffer is in the buffer subscribed state CCBs and the module triggers the slot start transition SLS at the

same time as the application locks the message buffer by the HL transition, the intermediate state is CCRx and the resulting state is locked buffer subscribed state HLckCCRx.

Table 26-104. Receive Message Buffer Transition Priorities

State	Priorities	Description
module vs. application		
Idle	BS > HD	Buffer Subscribed > Message Buffer Disable
HLck	BS > HD	Buffer Subscribed > Message Buffer Disable
CCRx	SSS > HL	Slot or Segment Start > Message Buffer Lock

26.6.6.3.2 Message Reception

As a result of the message buffer search, the controller changes the state of as many as two enabled receive message buffers from either idle state Idle or locked state HLck to the either subscribed state CCBs or locked buffer subscribed state HLckCCBs by triggering the buffer subscribed transition BS.

If the receive message buffers for the next slot are assigned to both channels, then at most one receive message buffer is changed to a buffer subscribed state.

If more than one matching message buffers assigned to a certain channel, then only the message buffer with the lowest message buffer number is in one of the states mentioned above.

With the start of the next static or dynamic slot the module trigger the slot start transition SLS. This changes the state of the subscribed receive message buffers from either CCBs to CCRx or from HLckCCBs to HLckCCRx, respectively.

During the reception slot, the received frame data are written into the shadow buffers. For details on receive shadow buffers, see Section 26.6.6.3.5, Receive Shadow Buffers Concept. The data and status of the receive message buffers that are the CCRx or HLckCCRx are not modified in the reception slot.

26.6.6.3.3 Message Buffer Update

With the start of the next static or dynamic slot or with the start of the symbol window or NIT, the module triggers the slot or segment start transition SSS. This transition changes the state of the receiving receive message buffers from either CCRx to CCSu or from HLckCCRx to HLck, respectively.

If a message buffer was in the locked state HLckCCRx, no update will be performed. The received data are lost. This is indicated by setting the Frame Lost Channel A/B Error Flag FRLA_EF/FRLB_EF in the CHI Error Flag Register (CHIERFR).

If a message buffer was in the CCRx state it is now in the CCSu state. After the evaluation of the slot status provided by the PE the message buffer is updated. The message buffer update depends on the slot status bits and the segment the message buffer is assigned to. This is described in Table 26-105.

Table 26-105. Receive Message Buffer Update

vSS!ValidFrame	*vRF!Header!NFIndicator*	Update description
1	1	Valid non-null frame received. - Message Buffer Data Field updated. - Frame Header Field updated. - Slot Status Field updated. - DUP:= 1 - DVAL:= 1 - MBIF:= 1
1	0	Valid null frame received. - Message Buffer Data Field *not* updated. - Frame Header Field *not* updated. - Slot Status Field updated. - DUP:= 0 - DVAL *not* changed - MBIF:= 1
0	x	No valid frame received. - Message Buffer Data Field not updated. - Frame Header Field not updated. - Slot Status Field updated. - DUP:= 0 - DVAL *not* changed. - MBIF:= 1, if the slot was not an empty dynamic slot. **Note:** An empty dynamic slot is indicated by the following frame and slot status bit values: *vSS!ValidFrame* = 0 and *vSS!SyntaxError* = 0 and *vSS!ContentError* = 0 and *vSS!BViolation* = 0.

NOTE

If the number of the last slot in the current communication cycle on a given channel is *n*, then all receive message buffers assigned to this channel with MBFIDR*n*[FID] > *n* will not be updated at all.

When the receive message buffer update has finished the status updated transition SU is triggered, which changes the buffer state from CCSu to Idle. An example receive message buffer timing and state change diagram for a normal frame reception is given in Figure 26-129.

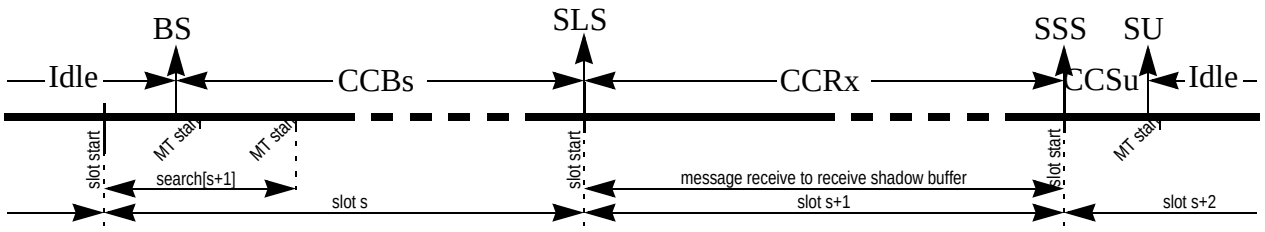


Figure 26-129. Message Reception Timing

The amount of message data written into the message buffer data field of the receive shadow buffer is determined by the following items:

1. The message buffer segment that the message buffer is assigned to, as defined by the Message Buffer Segment Size and Utilization Register (MBSSUTR).
2. The message buffer data field size, as defined by the related field of the Message Buffer Data Size Register (MBDSR).
3. The number of bytes received over the FlexRay bus.

If the message buffer is assigned to the message buffer segment 1, and the number of received bytes is greater than 2*MBDSR.MBSEG1DS, the controller writes only 2*MBDSR.MBSEG1DS bytes into the message buffer data field of the receive shadow buffer. If the number of received bytes is less than 2*MBDSR.MBSEG1DS, the controller writes only the received number of bytes and will not change the trailing bytes in the message buffer data field of the receive shadow buffer. The same holds for the message buffer segment 2 with MBDSR.MBSEG2DS.

26.6.6.3.4 Received Message Access

To access the message data received over the FlexRay bus, the application reads the message data stored in the message buffer data field of the corresponding receive message buffer. The access to the message buffer data field is described in Section 26.6.3.1, Individual Message Buffers.

The application can read the message buffer data field if the receive message buffer is one of the states HDis, HDisLck, or HLck. If the message buffer is in one of these states, the controller will not change the content of the message buffer.

26.6.6.3.5 Receive Shadow Buffers Concept

The receive shadow buffer concept applies only to individual receive message buffers. The intention of this concept is to ensure that only syntactically and semantically valid received non-null frames are presented to the application in a receive message buffer. The basic structure of a receive shadow buffer is described in Section 26.6.3.2, Receive Shadow Buffers.

The receive shadow buffers temporarily store the received frame header and message data. After the slot boundary the slot status information is generated. If the slot status information indicates the reception of the valid non-null frame (see Table 26-105), the controller writes the slot status into the slot status field of the receive shadow buffer and exchanges the content of the Message Buffer Index Registers (MBIDXRn) with the content of the corresponding internal shadow buffer index register. In all other cases, the controller writes the slot status into the identified receive message buffer, depending on the slot status and the FlexRay segment the message buffer is assigned to.

The shadow buffer concept, with its index exchange, results in the fact that the FlexRay memory located message buffer associated to an individual receive message buffer changes after successful reception of a valid frame. This means that the message buffer area in the FlexRay memory accessed by the application for reading the received message is different from the initial setting of the message buffer. Therefore, the application must not rely on the index information written initially into the Message Buffer Index Registers (MBIDXRn). Instead, the index of the message buffer header field must be fetched from the Message Buffer Index Registers (MBIDXRn).

26.6.6.4 Double Transmit Message Buffer

The section provides a detailed description of the functionality of the double transmit message buffers.

Double transmit message buffers are used by the application to provide the controller with the message data to be transmitted over the FlexRay Bus. The controller uses this message buffer to provide information to the application about the transmission process, and status information about the slot in which message data was transmitted.

In contrast to the single transmit message buffers, the application can provide new transmission data while the transmission of the previously provided message data is running. This scheme is called double buffering and can be considered as a FIFO of depth 2.

Double transmit message buffers are implemented by combining two individual message buffers that form the two sides of an double transmit message buffer. One side is called the *commit side* and will be accessed by the application to provide the message data. The other side is called the *transmit side* and is used by the controller to transmit the message data to the FlexRay bus. The two sides are located in adjacent individual message buffers. The message buffer that implements the commit side has an even message buffer number 2n. The transmit side message buffer follows the commit side message buffer and has the message buffer number 2n+1. The basic structure and data flow of a double transmit message buffer is given in Figure 26-130.

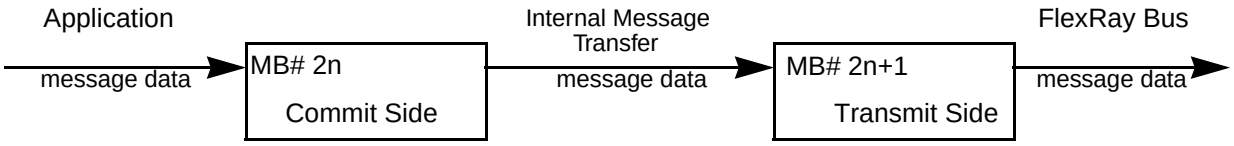


Figure 26-130. Double Transmit Buffer Structure and Data Flow

NOTE

Both the commit and the transmit side must be configured with identical values except for the Message Buffer Index Registers (MBIDXRn).

26.6.6.4.1 Access Regions

To certain message buffer fields, both the application and the controller have access. To ensure data consistency, a message buffer locking scheme is implemented, which controls the exclusive access to the data, control, and status bits of the message buffer.

The access scheme for double transmit message buffers is depicted in Figure 26-131. The given regions represent fields that can be accessed from both the application and the controller and, thus, require access restrictions. A description of the regions is given in Table 26-106.

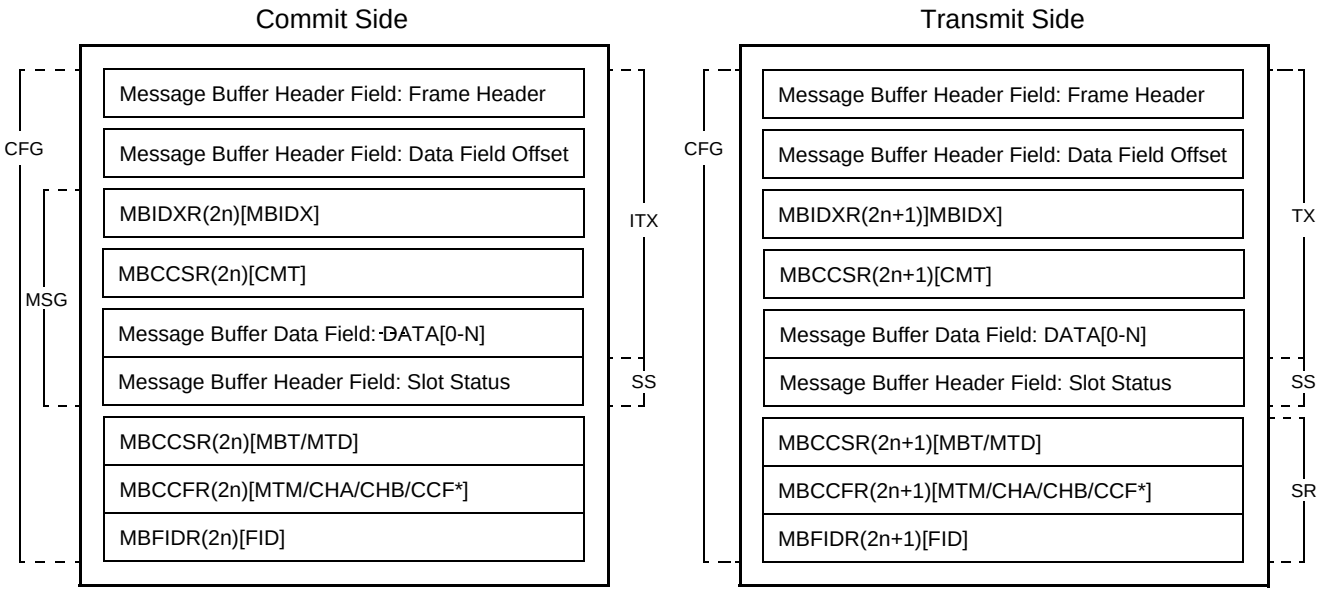


Figure 26-131. Double Transmit Message Buffer Access Regions Layout

Table 26-106. Double Transmit Message Buffer Access Regions Description

Access			Description
Region	Type		
	Application	Module	
Commit Side			
CFG	read/write	—	Message Buffer Configuration
MSG	read/write	—	Message Buffer Data and Control access
ITX	—	read/write	Internal Message Transfer.
SS	—	write-only	Slot Status Update
Transmit Side			
CFG	read/write	—	Message Buffer Configuration
SR	—	read-only	Message Buffer Search
TX	—	read-only	Internal Message Transfer, Message Transmission
SS	—	write-only	Slot Status Update

The trigger bits MBCCSR*n*[EDT] and MBCCSR*n*[LCKT], and the interrupt enable bit MBCCSR*n*[MBIE] are not under access control and can be accessed from the application at any time. The status bits MBCCSR*n*[EDS] and MBCCSR*n*[LCKS] are not under access control and can be accessed from the controller at any time.

The interrupt flag MBCCSR*n*.MBIF is not under access control and can be accessed from the application and the controller at any time. controller set access has higher priority.

The controller restricts its access to the regions, depending on the current state of the corresponding part of the double transmit message buffer. The application must adhere to these restrictions in order to ensure data consistency. The states for the commit side of a double transmit message buffer are given in Figure 26-132. A description of the states is given in Table 26-108. The states for the transmit side of a

double transmit message buffer are given in Figure 26-133. A description of the states is given in Table 26-108. The description tables also provide the access scheme for the access regions.

The status bits MBCCSR*n*[EDS] and MBCCSR*n*[LCKS] provide the application with the required message buffer status information. The internal status information is not visible to the application.

26.6.6.4.2 Message Buffer States

This section describes the transmit message buffer states and provides a state diagram.

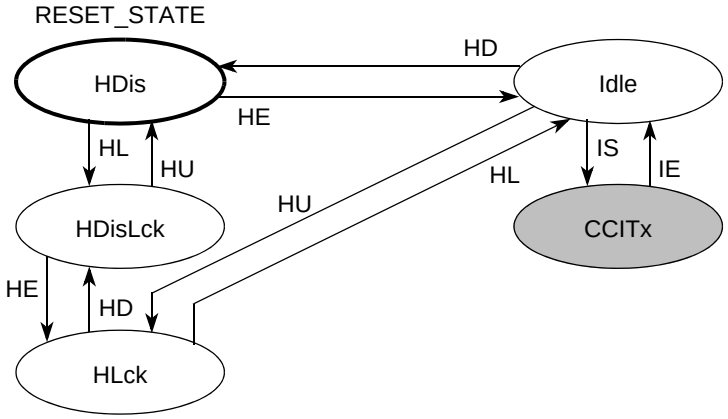


Figure 26-132. Double Transmit Message Buffer State Diagram (Commit Side)

A description of the states of the commit side of a double transmit message buffer is given in Table 26-107.

Table 26-107. Double Transmit Message Buffer State Description (Commit Side)

State	MBCCSR(2n)		Access Region		Description
	EDS	LCKS	Appl.	Module	
common states					
HDis	0	0	CFG	–	**Disabled** - Message Buffer under configuration. Commit Side can *not* be used for internal message transfer.
CCITx	1	0	–	ITX	**Internal Message Transfer** - Message Buffer Data transferred from commit side to transmit side.
commit side specific states					
Idle	1	0	–	ITX, SS	**Idle** - Message Buffer Commit Side is idle. Commit Side can be used for internal message transfer.
HDisLck	0	1	CFG	SS	**Disabled and Locked** - Message Buffer under configuration. Commit Side can *not* be used for internal message transfer.
HLck	1	1	MSG	SS	Locked - Applications access to data, control, and status. Commit Side can *not* be used for internal message transfer.

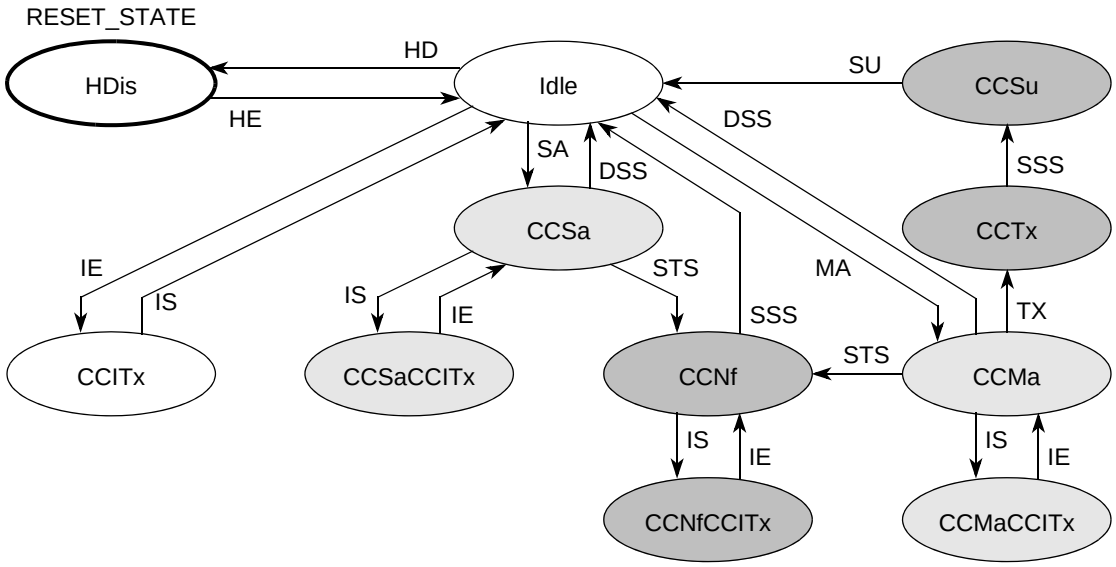


Figure 26-133. Double Transmit Message Buffer State Diagram (Transmit Side)

A description of the states of the transmit side of a double transmit message buffer is given in Table 26-108.

Table 26-108. Double Transmit Message Buffer State Description (Transmit Side)

State	MBCCSR*n*		Access Region		Description
	EDS	LCKS	Appl.	Module	
common states					
HDis	0	0	CFG	–	**Disabled** - Message Buffer under configuration. Excluded from message buffer search.
CCITx	1	0	–	TX	**Internal Message Transfer** - Message Buffer Data transferred from commit side to transmit side.
transmit side specific states					
Idle	1	0	–	SR	**Idle** - Message Buffer Transmit Side is idle. Transmit Side is included in message buffer search.
CCSa	1	0	–	–	Slot Assigned - Message buffer assigned to next static slot. Ready for Null Frame transmission.
CCSaCCITx	1	0	–	TX	Slot Assigned and Internal Message Transfer - Message buffer assigned to next static slot and Message Buffer Data transferred from commit side to transmit side.
CCNf	1	0	–	TX	Null Frame Transmission Header is used for null frame transmission.
CCNfCCITx	1	0	–	TX	Null Frame Transmission and Internal Message Transfer - Header is used for null frame transmission and Message Buffer Data transferred from commit side to transmit side.
CCMa	1	0	–	–	Message Available - Message buffer is assigned to next slot and cycle counter filter matches.
CCMaCCITx	1	0	–	–	Message Available and Internal Message Transfer - Message buffer is assigned to next slot and cycle counter filter matches and Message Buffer Data transferred from commit side to transmit side.

Table 26-108. Double Transmit Message Buffer State Description (Transmit Side) (continued)

State	MBCCSR*n*		Access Region		Description
	EDS	LCKS	Appl.	Module	
CCTx	1	0	–	TX	Message Transmission - Message buffer data transmit. Payload data from buffer transmitted
CCSu	1	0	–	SS	Status Update - Message buffer status update. Update of status flags, the slot status field, and the header index. Note: The slot status field of the commit side is updated too, even if the application has locked the commit side.

26.6.6.4.3 Message Buffer Transitions

Application Transitions

The application transitions that can be triggered by the application using the commands described in Table 26-109. The application issues the commands by writing to the Message Buffer Configuration, Control, Status Registers (MBCCSRn). Only one command can be issued with one write access. Each command is executed immediately. If the command is ignored, it must be issued again.

Message Buffer Enable and Disable

The enable and disable commands can be issued on the transmit side only. Any enable or disable command issued on the commit side will be ignored without notification. The transitions that will be triggered depends on the value of the EDS bit. The enable and disable commands will affect both the commit side and the transmit side at the same time. If the application triggers the disable transition HD while the transmit side is in one of the states CCSa, CCSaCCITx, CCNf, CCNfCCITx, CCMa, CCMaCCITx, CCTx, or CCSu, the disable transition has no effect (command is ignored) and the message buffer state is not changed. No notification is given to the application.

Message Buffer Lock and Unlock

The lock and unlock commands can be issued on the commit side only. Any lock or unlock command issued on the transmit side will be ignored and the double transmit buffer lock error flag DBL_EF in the CHI Error Flag Register (CHIERFR) will be set. The transitions that will be triggered depends on the current value of the LCKS bit. The lock and unlock commands will only affect the commit side. If the application triggers the lock transition HL while the commit side is in the state CCITx, the message buffer state will not be changed and the message buffer lock error flag LCK_EF in the CHI Error Flag Register (CHIERFR) will be set.

Table 26-109. Double Transmit Message Buffer Host Transitions

Transition	Host Command	Condition	Description
HE	MBCCSR(2*n* + 1)[EDT] = 1	MBCCSR(2*n* + 1)[EDS] = 0	Application triggers message buffer enable.
HD		MBCCSR(2*n*+1)[EDS] = 1	Application triggers message buffer disable.
HL	MBCCSR(2*n*)[LCKT] = 1	MBCCSR(2*n*)[LCKS] = 0	Application triggers message buffer lock.
HU		MBCCSR(2*n*)[LCKS] = 1	Application triggers message buffer unlock.

Module Transitions

The module transitions that can be triggered by the controller are described in Table 26-110. The transitions C1 and C2 apply to both sides of the message buffer and are applied at the same time. All other controller transitions apply to the transmit side only.

Table 26-110. Double Transmit Message Buffer Module Transitions

Transition	Condition	Description
common transitions		
IS	see Section 26.6.6.4.5, Internal Message Transfer	Internal Message Transfer Start - Start transfer of message data from commit side to transmit side.
IE		Internal Message Transfer End - Stop transfer of message data from commit side to transmit side. Note: The internal message transfer is stopped before the slot or segment start.
transmit side specific transitions		
SA	slot match and static slot	Slot Assigned - Message buffer is assigned to next static slot.
MA	slot match and CycleCounter match	Message Available - Message buffer is assigned to next slot and cycle counter filter matches.
TX	slot start and MBCCSR(2*n* + 1)[CMT] = 1	Transmission Slot Start - Slot Start and commit bit CMT is set. In case of a dynamic slot, pLatestTx is not exceeded.
SU	status updated	Status Updated - Slot Status field and message buffer status flags updated. Interrupt flag set.
STS	static slot start	Static Slot Start - Start of static slot.
DSS	dynamic slot start or symbol window start or NIT start	Dynamic Slot or Segment Start. - Start of dynamic slot or symbol window or NIT.
SSS	slot start or symbol window start or NIT start	Slot or Segment Start - Start of static slot or dynamic slot or symbol window or NIT.

Transition Priorities

The application can trigger only one transition at a time. There is no need to specify priorities among them.

As shown in the first part of Table 26-111, the module transitions have a higher priority than the application transitions. The priorities among the controller transitions and the related states are given in the second part of Table 26-111. These priorities apply only to the transmit side. The internal message transmit start transition IS has tho lowest priority.

Table 26-111. Double Transmit Message Buffer Transition Priorities

State	Priority	Description
module vs. application		
Idle	IS > HD IS > HL	Internal Message Transfer Start > Message Buffer Disable Internal Message Transfer Start > Message Buffer Lock
module internal		
Idle	MA > SA	Message Available > Slot Assigned

Table 26-111. Double Transmit Message Buffer Transition Priorities (continued)

State	Priority	Description
CCMa	TX > STS TX > DSS	Transmission Slot Start > Static Slot Start Transmission Slot Start > Dynamic Slot Start

26.6.6.4.4 Message Preparation

The application provides the message data through the commit side. The transmission itself is executed from the transmit side. The transfer of the message data from the commit side to the transmit side is done by the *Internal Message Transfer*, which is described in Section 26.6.6.4.5, Internal Message Transfer.

To transmit a message over the FlexRay bus, the application writes the message data into the message buffer data field of the commit side and sets the commit bit CMT in the Message Buffer Configuration, Control, Status Registers (MBCCSRn). The physical access to the message buffer data field is described in Section 26.6.3.1, Individual Message Buffers.

As indicated by Table 26-107, the application shall write to the message buffer data field and change the commit bit CMT only if the transmit message buffer is in one of the states HDis, HDisLck, or HLck. The application can change the state of a message buffer if it issues the appropriate commands shown in Table 26-109. The state change is indicated through the MBCCSR*n*[EDS] and MBCCSR*n*[LCKS] status bits.

26.6.6.4.5 Internal Message Transfer

The internal message transfer transfers the message data from the commit side to the transmit side. The internal message transfer is implemented as the swapping of the content of the Message Buffer Index Registers (MBIDXRn) of the commit side and the transmit side. After the swapping, the commit side CMT bit is cleared, the commit side interrupt flag MBIF is set, the transmit side CMT bit is set, and the transmit side DVAL bit is cleared.

The conditions and the point in time when the internal message transfer is started are controlled by the message buffer commit mode bit MCM in the Message Buffer Configuration, Control, Status Registers (MBCCSRn). The MCM bit configures the message buffer for either the streaming commit mode or the immediate commit mode. A detailed description is given in Streaming Commit Mode and Immediate Commit Mode. The Internal Message Transfer is triggered with the transition IS. Both sides of the message buffer enter one of the CCITx states. The internal message transfer is finished with the transition IE.

Streaming Commit Mode

The intention of the streaming commit mode is to ensure that each committed message is transmitted *at least once*. The controller will not start the Internal Message Transfer for a message buffer as long as the message data on the transmit side is not transmitted at least once.

The streaming commit mode is configured by clearing the message buffer commit mode bit MCM in the Message Buffer Configuration, Control, Status Registers (MBCCSRn).

In this mode, the internal message transfer from the commit side to the transmit side is started for a double transmit message buffer when all of the following conditions are fulfilled

1. the commit side is in the Idle state
2. the commit site message data are valid (MBCCSR(2*n*)[CMT] = 1)
3. the transmit side is in one of the states Idle, CCSa, or CCMa
4. the transmit side contains either no valid message data (MBCCSR(2*n* + 1)[CMT] = 0) or the message data were transmitted at least once (MBCCSR(2*n* + 1)[DVAL] = 1)

An example of a streaming commit mode state change diagram is given in Figure 26-134. In this example, both the commit and the transmit side do not contain valid message data and the application provides two messages. The message buffer does not match the next slot.

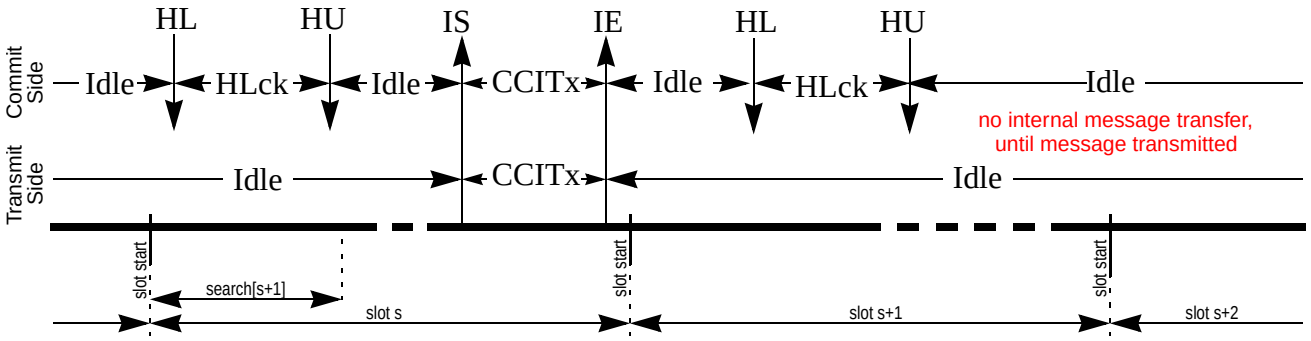


Figure 26-134. Internal Message Transfer in Streaming Commit Mode

Immediate Commit Mode

The intention of the immediate commit mode is to transmit the *latest* data provided by the application. This implies that it is not guaranteed that each provided message will be transmitted at least once.

The immediate commit mode is configured by setting the message buffer commit mode bit MCM in the Message Buffer Configuration, Control, Status Registers (MBCCSRn).

In this mode, the internal message transfer from the commit side to the transmit side is started for one double transmit message buffer when all of the following conditions are fulfilled

1. The commit side is in the Idle state.
2. The commit site message data are valid (MBCCSR(2*n*)[CMT] = 1).
3. The transmit side is in one of the states Idle, CCSa, or CCMa.

It is not checked whether the transmit side contains no valid message data or valid message data were transmitted at least once. If message data are valid and not transmitted, they may be overwritten.

An example of a streaming commit mode state change diagram is given in Figure 26-135. In this example, both the commit and the transmit side do not contain valid message data, and the application provides two messages and the first message is gets overwritten. The message buffer does not match the next slot.

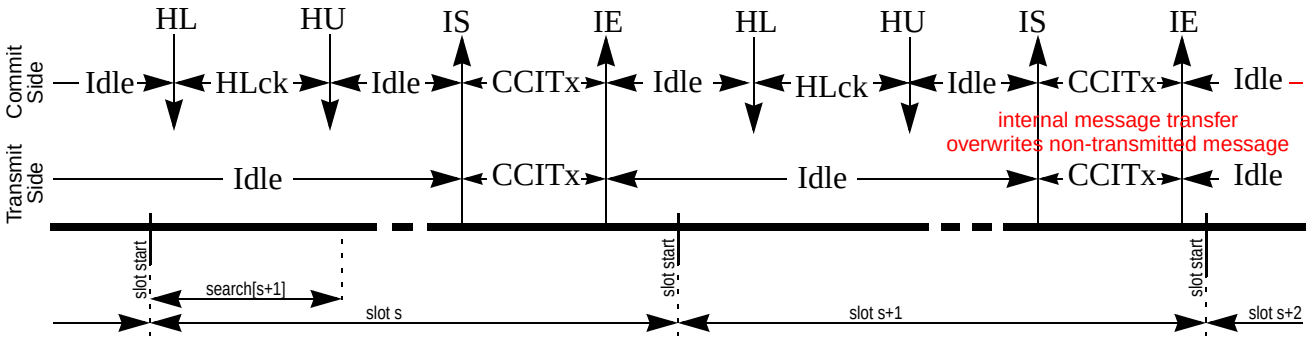


Figure 26-135. Internal Message Transfer in Immediate Commit Mode

26.6.6.4.6 Message Transmission

For double transmit message buffers, the message buffer search checks only the transmit side part. The internal scheduling ensures, that the internal message transfer is stopped on the message buffer search start. Thus, the transmit side of message buffer, that is not in its transmission or status update slot, is always in the Idle state.

The message transmit behavior and transmission state changes of the transmit side of a double transmit message buffer are the same as for single buffered transmit buffers, except that the transmit side of double buffers cannot be locked by the application, that is, the HU and HL transition do not exist. Therefore, refer to Section 26.6.6.2.5, Message Transmission.

26.6.6.4.7 Message Buffer Status Update

The message buffer status update behavior of the transmit side of a double transmit message buffer is the same as for single transmit message buffers which is described in Section 26.6.6.2.7, Message Buffer Status Update.

Additionally, the slot status field of the commit side is update after the update of the slot status field of the transmit side, even if the commit side is locked by the application. This is implemented to provide the slot status of the most recent transmission slot.

26.6.7 Individual Message Buffer Search

This section provides a detailed description of the message buffer search algorithm.

The message buffer search determines for each enabled channel if a slot *s* in a communication cycle *c* is assigned for frame or null frame transmission or if it is subscribed for frame reception on that channel.

The message buffer search is a sequential algorithm which is invoked at the following protocol related events:

1. NIT start
2. slot start in the static segment
3. minislot start in the dynamic segment

The message buffer search within the NIT searches for message buffers assigned or subscribed to slot 1. The message buffer search within slot *n* searches for message buffers assigned or subscribed to slot *n+1*.

In general, the message buffer search for the next slot *n* considers only message buffers which are

1. enable (MBCCSR*n*[EDS] = 1), and
2. matches the next slot *n* (MBFIDR*n*[FID] = *n*), and
3. are the transmit side buffer in case of a double transmit message buffer.

On top of that, for the static segment only those message buffers are considered, that match the condition of at least one row of Table 26-112. For the dynamic segment only those message buffers are considered, that match the condition of at least one row of Table 26-113. These message buffers are called *matching* message buffers.

For each enabled channel the message buffer search may identify multiple *matching* message buffers. Among all matching message buffers the message buffers with highest priority according to Table 26-112 for the static segment and according to Table 26-113 for the dynamic segment are selected.

Table 26-112. Message Buffer Search Priority (static segment)

Priority	MTD	LCKS	CMT	CCFM[1]	Description	Transition
(highest) 0	1	0	1	1	transmit buffer, matches cycle count, not locked and committed	MA
1	1	-	0	1	transmit buffer, matches cycle count, not committed	SA
	1	1	-	1	transmit buffer, matches cycle count, locked	SA
2	1	—	—	—	transmit buffer	SA
3	0	0	n/a	1	receive buffer, matches cycle count, not locked	SB
(lowest) 4	0	1	n/a	1	receive buffer, matches cycle count, locked	SB

1 Cycle Counter Filter Match, see Section 26.6.7.1, Message Buffer Cycle Counter Filtering.

Table 26-113. Message Buffer Search Priority (dynamic segment)

Priority	MTD	LCKS	CMT	CCFM[1]	Description	Transition
(highest) 0	1	0	1	1	transmit buffer, matches cycle count, not locked and committed	MA
1	0	0	n/a	1	receive buffer, matches cycle count, not locked	SB
(lowest) 2	0	1	n/a	1	receive buffer, matches cycle count, locked	SB

1 Cycle Counter Filter Match, see Section 26.6.7.1, Message Buffer Cycle Counter Filtering.

If there are multiple message buffer with highest priority, the message buffer with the lowest message buffer number is selected. All message buffer which have the highest priority must have a consistent channel assignment as specified in Section 26.6.7.2, Message Buffer Channel Assignment Consistency.

Depending on the message buffer channel assignment the same message buffer can be found for both channel A and channel B. In this case, this message buffer is used as described in Section 26.6.3.1, Individual Message Buffers.

26.6.7.1 Message Buffer Cycle Counter Filtering

The message buffer cycle counter filter is a value-mask filter defined by the CCFE, CCFMSK, and CCFVAL fields in the Message Buffer Cycle Counter Filter Registers (MBCCFRn). This filter determines a set of communication cycles in which the message buffer is considered for message reception or message transmission. If the cycle counter filter is disabled (CCFE = 0), this set of cycles consists of all communication cycles.

If the cycle counter filter of a message buffer does not match a certain communication cycle number, this message buffer is not considered for message transmission or reception in that communication cycle. In case of a transmit message buffer assigned to a slot in the static segment, though, this buffer is added to the matching message buffers to indicate the slot assignment and to trigger the null frame transmission.

The cycle counter filter of a message buffer matches the communication cycle with the number CYCCNT if at least one of the following conditions evaluates to true:

$$\text{MBCCFRn[CCFE]} = 0 \qquad \textit{Eqn. 26-12}$$

$$\text{CYCCNT \& MBCCFRn[CCFMSK]} = \text{MBCCFRn[CCFVAL] \& MBCCFRn[CCFMSK}} \qquad \textit{Eqn. 26-13}$$

26.6.7.2 Message Buffer Channel Assignment Consistency

The message buffer channel assignment given by the CHA and CHB bits in the Message Buffer Cycle Counter Filter Registers (MBCCFRn) defines the channels on which the message buffer will receive or transmit. The message buffer with number *n* transmits or receives on channel A if MBCCFR*n*[CHA] = 1 and transmits or receives on channel B if MBCCFR*n*[CHB] = 1.

To ensure correct message buffer operation, all message buffers assigned to the same slot and with the same priority must have a *consistent* channel assignment. That means they must be either assigned to one channel only, or must be assigned to *both* channels. The behavior of the message buffer search is not defined, if both types of channel assignments occur for one slot and priority. An inconsistent channel assignment for message buffer 0 and message buffer 1 is depicted in Figure 26-136.

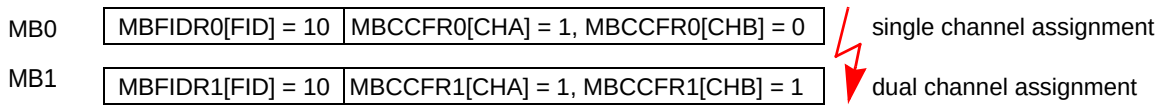


Figure 26-136. Inconsistent Channel Assignment

26.6.7.3 Node Related Slot Multiplexing

The term *Node Related Slot Multiplexing* applies to the dynamic segment only and refers to the functionality if there are transmit as well as receive message buffers are configured for the same slot.

According to Table 26-113 the transmit buffer is only found if the cycle counter filter matches, and the buffer is not locked and committed. In all other cases, the receive buffer will be found. Thus, if the block has no data to transmit in a dynamic slot, it is able to receive frames on that slot.

26.6.7.4 Message Buffer Search Error

If the message buffer search is running while the next message buffer search start event appears, the message buffer search is stopped and the Message Buffer Search Error Flag MSB_EF is set in the CHI Error Flag Register (CHIERFR). This appears only if the CHI frequency is too low to search through all message buffers within the NIT or a minislot. The message buffer result is not defined in this case. For more details see Section 26.7.3, Number of Usable Message Buffers.

26.6.8 Individual Message Buffer Reconfiguration

The initial configuration of each individual message buffer can be changed even when the protocol is not in the *POC:config* state. This is referred to as individual message buffer *reconfiguration*. The configuration bits and fields that can be changed are given in the section on Specific Configuration Data. The common configuration data given in the section on Specific Configuration Data cannot be reconfigured when the protocol is out of the *POC:config* state.

26.6.8.1 Reconfiguration Schemes

Depending on the target and destination basic state of the message buffer that is to be reconfigured, there are three reconfiguration schemes.

26.6.8.1.1 Basic Type Not Changed (RC1)

A reconfiguration will not change the basic type of the individual message buffer, if both the message buffer transfer direction bit MBCCSR*n*[MTD] and the message buffer type bit MBCCSR*n*[MBT] are not changed. This type of reconfiguration is denoted by RC1 in Figure 26-137. Single transmit and receive message buffers can be RC1-reconfigured when in the HDis or HDisLck state. Double transmit message buffers can be RC1-reconfigured if both the transmit side and the commit side are in the HDis state.

26.6.8.1.2 Buffer Type Not Changed (RC2)

A reconfiguration will not change the buffer type of the individual message buffer if the message buffer buffer type bit MBCCSR*n*[MBT] is not changed. This type of reconfiguration is denoted by RC2 in Figure 26-137. It applies only to single transmit and receive message buffers. Single transmit and receive message buffers can be RC2-reconfigured when in the HDis or HDisLck state.

26.6.8.1.3 Buffer Type Changed (RC3)

A reconfiguration will change the buffer type of the individual message buffer if the message buffer type bit MBCCSR*n*[MBT] is changed. This type of reconfiguration is denoted by RC3 in Figure 26-137. The RC3 reconfiguration splits one double buffer into two single buffers or combines two single buffer into one double buffer. In the later case, the two single message buffers must have consecutive message buffer numbers and the smaller one must be even. Message Buffers can be RC3 reconfigured if they are in the HDis state.

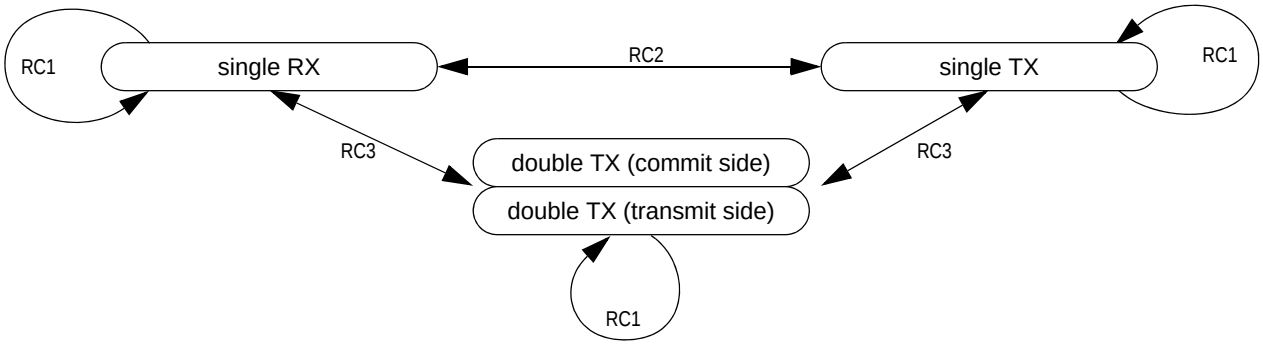


Figure 26-137. Message Buffer Reconfiguration Scheme

26.6.9 Receive FIFOs

This section provides the functional description of the two receive FIFOs.

26.6.9.1 Overview

The two receive FIFOs implement the queued message buffer concept defined by the *FlexRay Communications System Protocol Specification, Version 2.1 Rev A.* One FIFO is assigned to channel A, the other FIFO is assigned to channel B. Both FIFOs work completely independent from each other.

The message buffer structure of each FIFO is described in Section 26.6.3.3, Receive FIFO. The area in the FlexRay memory for each of the two FIFOs is characterized by:

- The FIFO system memory base address
- The index of the first FIFO entry given by Receive FIFO Start Index Register (RFSIR)
- The number of FIFO entries and the length of each FIFO entry as given by Receive FIFO Depth and Size Register (RFDSR)

26.6.9.2 FIFO Configuration

The FIFOs can be configured for two different locations of the system memory base address via the FIFO address mode bit FAM in the Module Configuration Register (MCR).

26.6.9.2.1 Single System Memory Base Address Mode

This mode is configured, when the FIFO address mode flag MCR[FAM] is set to 0. In this mode, the location of the system memory base address for the FIFO buffers is System Memory Base Address Register (SYMBADR).

26.6.9.2.2 Dual System Memory Base Address Mode

This mode is configured, when the FIFO address mode flag MCR[FAM] is set to 1. In this mode, the location of the system memory base address for the FIFO buffers is Receive FIFO System Memory Base Address Register (RFSYMBADR).

The FIFO control and configuration data are given in Section 26.6.3.7, Receive FIFO Control and Configuration Data. The configuration of the FIFOs consists of two steps.

The first step is the allocation of the required amount of FlexRay memory for the FlexRay window. This includes the allocation of the message buffer header area and the allocation of the message buffer data fields. For more details see Section 26.6.4, FlexRay Memory Layout.

The second step is the programming of the configuration data register while the PE is in *POC:config*.

The following steps configure the layout of the FIFO.

- Configure the FIFO update and address modes in Module Configuration Register (MCR)
- Configure the FIFO system memory base address
- Configure the Receive FIFO Start Index Register (RFSIR) with the first message buffer header index that belongs to the FIFO
- Configure the Receive FIFO Depth and Size Register (RFDSR) with FIFO entry size
- Configure the Receive FIFO Depth and Size Register (RFDSR) with FIFO depth
- Configure the FIFO Filters

26.6.9.3 FIFO Periodic Timer

The FIFO periodic timer is used to generate an FIFO almost-full interrupt at certain point in time, if the almost-full watermark is not reached, but the FIFO is not empty. This can be used to prevent frames from get stuck in the FIFO for a long time.

The FIFO periodic timer is configured via the Receive FIFO Periodic Timer Register (RFPTR). If the periodic timer duration RFPTIR[PTD] is configured to 0x0000, the periodic timer is continuously expired. If the periodic timer duration RFPTIR[PTD] is configured to 0x3FFF, the periodic timer never expires. If the periodic timer is configured to a value *ptd*, greater than 0x0000 and smaller 0x3FFF, the periodic timer expires and is restarted at the start of every communication cycle, and expires and is restarted after *ptd* macroticks have been elapsed.

26.6.9.4 FIFO Reception

The FIFO reception is a controller internal operation.

A message frame reception is directed into the FIFO, if no individual message buffer is assigned for transmission or subscribed for reception for the current slot. In this case the FIFO filter path shown in Figure 26-138 is activated.

If the FIFO filter path indicates that the received frame has to be appended to the FIFO and the FIFO is not full, the controller writes the received frame header into the message buffer header field indicated by the controller internal FIFO write index. The frame payload data are written into the corresponding message buffer data field. If the status of the received frame indicates a valid non-null frame, the slot status information is written into the message buffer header field and the controller internal FIFO write index is updated by 1 and the fifo fill level FLA (FLB) in the Receive FIFO Fill Level and POP Count Register (RFFLPCR) is incremented.If the status of the received frame indicates an invalid or null frame, the frame is not appended to the FIFO.

26.6.9.5 FIFO Almost-Full Interrupt Generation

If the fifo fill level FLA (FLB) is updated after a frame reception and exceeds the FIFO watermark level WM, i.e. FLA>WM_A (FLB>WM_B), then the FIFO almost-full interrupt flag GIFER[FAFAIF] (GIFER[FAFBIF]) is asserted.If the periodic timer expires, and FIFOA (FIFOB) is not empty, i.e. FLA>0 (FLB>0), then the FIFO almost-full interrupt flag GIFER[FAFAIF] (GIFER[FAFBIF]) is asserted.

26.6.9.6 FIFO Overflow Error Generation

If the FIFOA (FIFOB) is full, i.e. FLA=$FIFO\_DEPTH_A$ (FLB=$FIFO\_DEPTH_B$) and the conditions for a FIFO reception as described in Section 26.6.9.4, FIFO Reception, are fulfilled, then the fifo overflow error flag CHIERFR[FOVA_EF] (CHIERFR[FOVB_EF]) is asserted.

26.6.9.7 FIFO Message Access

The FIFOA (FIFOB) contains valid messages if the FIFO fill level FLA (FLB) is greater than 0. The Receive FIFO A Read Index Register (RFARIR) (Receive FIFO B Read Index Register (RFBRIR)) pointing to a message buffer with valid content and the oldest frames stored in the FIFO.

If the FIFO fill level FLA (FLB) is 0, than the FIFOA (FIFOB) contains no valid messages and the Receive FIFO A Read Index Register (RFARIR) (Receive FIFO B Read Index Register (RFBRIR)) pointing to a message buffer with invalid content. In this case the application must not read data from the FIFO.

To access the oldest message in the FIFOA (FIFOB), the application first reads the read index RDIDX out of the Receive FIFO A Read Index Register (RFARIR) (Receive FIFO B Read Index Register (RFBRIR)). This read index points to the message buffer header field of the oldest message buffer that contains valid received message data. The application can access the message data as described in Section 26.6.3.3, Receive FIFO. When the application has read the message buffer data and status information, it can update the FIFO as described in Section 26.6.9.8, FIFO Update.

26.6.9.8 FIFO Update

The application updates the FIFOA (FIFOB) by writing a pop count value *pc* different from 0 to the PCA (PCB) field in the Receive FIFO Fill Level and POP Count Register (RFFLPCR).

As a result of the this operation, the controller removes the oldest *pc* entries from FIFOA (FIFOB).

If the specified pop count value *pc* is greater than the current fill level *fl* provided in FLA (FAB) field, then only *fl* entries are removed from the FIFOA (FIFOB), the remaining *fl-pc* requested pop operations are discarded without any notification. In this case FIFOA (FIFOB) is empty after the update operation.

The read index in the Receive FIFO A Read Index Register (RFARIR) (Receive FIFO B Read Index Register (RFBRIR)) is incremented by the number of removed items. If the read index reaches the top of the FIFO, it wraps around to the FIFO start index defined in Receive FIFO Start Index Register (RFSIR) automatically.

26.6.9.8.1 FIFO Interrupt Flag Update

Th FIFO Interrupt Flag Update mode is configured, when the FIFO update mode flag MCR[FUM] is set to 0. In this mode FIFOA (FIFOB) will be updated by 1 entry, when the interrupt flag GIFER[FAFAIF] (GIFER[FAFBIF]) is written with 1 by the application.

If the FIFO is empty, the update request is ignored without any notification.

The read index in the Receive FIFO A Read Index Register (RFARIR) (Receive FIFO B Read Index Register (RFBRIR)) is incremented by 1, if the FIFO was not empty. If the read index reaches the top of the FIFO, it wraps around to the FIFO start index automatically.

26.6.9.9 FIFO Filtering

The FIFO filtering is activated after all enabled individual receive message buffers have been searched without success for a message buffer to receive the current frame.

The controller provides three sets of FIFO filters. The FIFO filters are applied to valid non-null frames only. The FIFO will not receive invalid or null-frames. For each FIFO filter, the pass criteria is specified in the related section given below. Only frames that have passed all filters will be appended to the FIFO. The FIFO filter path is depicted in Figure 26-138.

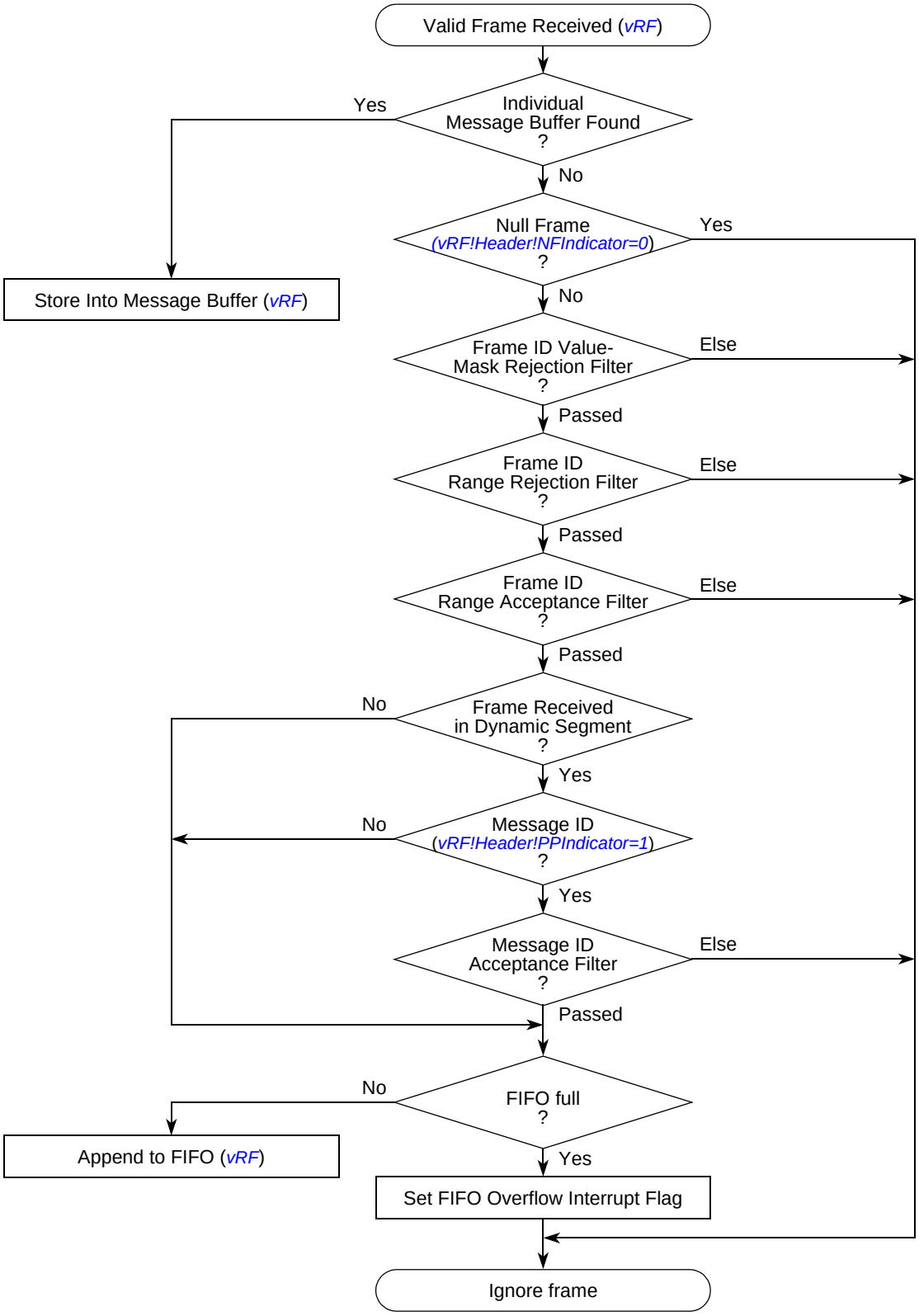


Figure 26-138. Received Frame FIFO Filter Path

A received frame passes the FIFO filtering if it has passed all three type of filter.

26.6.9.9.1 RX FIFO Frame ID Value-Mask Rejection Filter

The frame ID value-mask rejection filter is a value-mask filter and is defined by the fields in the Receive FIFO Frame ID Rejection Filter Value Register (RFFIDRFVR) and the Receive FIFO Frame ID Rejection Filter Mask Register (RFFIDRFMR). Each received frame with a frame ID FID that does not match the value-mask filter value passes the filter, i.e. is not rejected.

Consequently, a received valid frame with the frame ID FID passes the RX FIFO Frame ID Value-Mask Rejection Filter if Equation 26-14 is fulfilled.

$$\text{ID} \,\&\, \text{RFFIDRFMR[FIDRFMSK]} \neq \text{RFFIDRFVR[FIDRFVAL]} \,\&\, \text{RFFIDRFMR[FIDRFMSK} \qquad \textit{Eqn. 26-14}$$

The RX FIFO Frame ID Value-Mask Rejection Filter can be configured to pass all frames by the following settings.

- RFFIDRFVR[FIDRFVAL]:= 0x000 and RFFIDRFMR[FIDRFMSK]:= 0x7FF

Using the settings above, only the frame with frame ID 0 will be rejected, which is an invalid frame. All other frames will pass.

The RX FIFO Frame ID Value-Mask Rejection Filter can be configured to reject all frames by the following settings.

- RFFIDRFMR[FIDRFMSK]:= 0x000

Using the settings above, Equation 26-14 can never be fulfilled (0!= 0) and thus all frames are rejected; no frame will pass. This is the reset value for the RX FIFO.

26.6.9.9.2 RX FIFO Frame ID Range Rejection Filter

Each of the four RX FIFO Frame ID Range filters can be configured as a rejection filter. The filters are configured by the Receive FIFO Range Filter Configuration Register (RFRFCFR) and controlled by the Receive FIFO Range Filter Control Register (RFRFCTR). The RX FIFO Frame ID range filters apply to all received valid frames. A received frame with the frame ID FID passes the RX FIFO Frame ID Range rejection filters if either no rejection filter is enabled, or, for all of the enabled RX FIFO Frame ID Range rejection filters, i.e. RFRFCTR.FiMD = 1 and RFRFCTR.FiEN = 1, Equation 26-15 is fulfilled.

$$\text{FID} < \text{RFRFCFR}_{\text{SEL}}[\text{SID}_{\text{IBD}=0}]) \text{ or } (\text{RFRFCFR}_{\text{SEL}}[\text{SID}_{\text{IBD}=1}] < \text{FID}) \qquad \textit{Eqn. 26-15}$$

Consequently, all frames with a frame ID that fulfills Equation 26-16 for at least one of the enabled rejection filters will be rejected and thus not pass.

$$\text{RFRFCFR}_{\text{SEL}}[\text{SID}_{\text{IBD}=0}] \leq \text{FID} \leq \text{RFRFCFR}_{\text{SEL}}[\text{SID}_{\text{IBD}=1}] \qquad \textit{Eqn. 26-16}$$

26.6.9.9.3 RX FIFO Frame ID Range Acceptance filter

Each of the four RX FIFO Frame ID Range filters can be configured as an acceptance filter. The filters are configured by the Receive FIFO Range Filter Configuration Register (RFRFCFR) and controlled by the Receive FIFO Range Filter Control Register (RFRFCTR). The RX FIFO Frame ID range filters apply to all received valid frames. A received frame with the frame ID FID passes the RX FIFO Frame ID Range

acceptance filters if either no acceptance filter is enabled, or, for at least one of the enabled RX FIFO Frame ID Range acceptance filters, i.e. RFRFCTR.FiMD = 0 and RFRFCTR.FiEN = 1, Equation 26-17 is fulfilled.

$$RFRFCFR_{SEL}[SID_{IBD=0}] \le FID \le RFRFCFR_{SEL}[SID_{IBD=1}] \quad \textit{Eqn. 26-17}$$

26.6.9.9.4 RX FIFO Message ID Acceptance Filter

The RX FIFO Message ID Acceptance Filter is a value-mask filter and is defined by the Receive FIFO Message ID Acceptance Filter Value Register (RFMIDAFVR) and the Receive FIFO Message ID Acceptance Filter Mask Register (RFMIAFMR). This filter applies only to valid frames received in the dynamic segment with the payload preamble indicator bit PPI set to 1. All other frames will pass this filter.

A received valid frame in the dynamic segment with the payload preamble indicator bit PPI set to 1 and with the message ID MID (the first two bytes of the payload) will pass the RX FIFO Message ID Acceptance Filter if Equation 26-18 is fulfilled.

$$MID \,\&\, RFMIDAFMR[MIDAFMSK] = RFMIDAFMR[MIDAFVAL] \,\&\, RFMIDAFMR[MIDAFMSK] \quad \textit{Eqn. 26-18}$$

The RX FIFO Message ID Acceptance Filter can be configured to accept all frames by setting

- RFMIDAFMR[MIDAFMSK]:= 0x000

Using the settings above, Equation 26-18 is always fulfilled and all frames will pass.

26.6.10 Channel Device Modes

This section describes the two FlexRay channel device modes that are supported by the controller.

26.6.10.1 Dual Channel Device Mode

In the dual channel device mode, both FlexRay ports are connected to physical FlexRay bus lines. The FlexRay port consisting of FR_A_RX, FR_A_TX, and $\overline{\text{FR\_A\_TX\_EN}}$ is connected to the physical bus channel A and the FlexRay port consisting of FR_B_RX, FR_B_TX, and $\overline{\text{FR\_B\_TX\_EN}}$ is connected to the physical bus channel B. The dual channel system is shown in Figure 26-139.

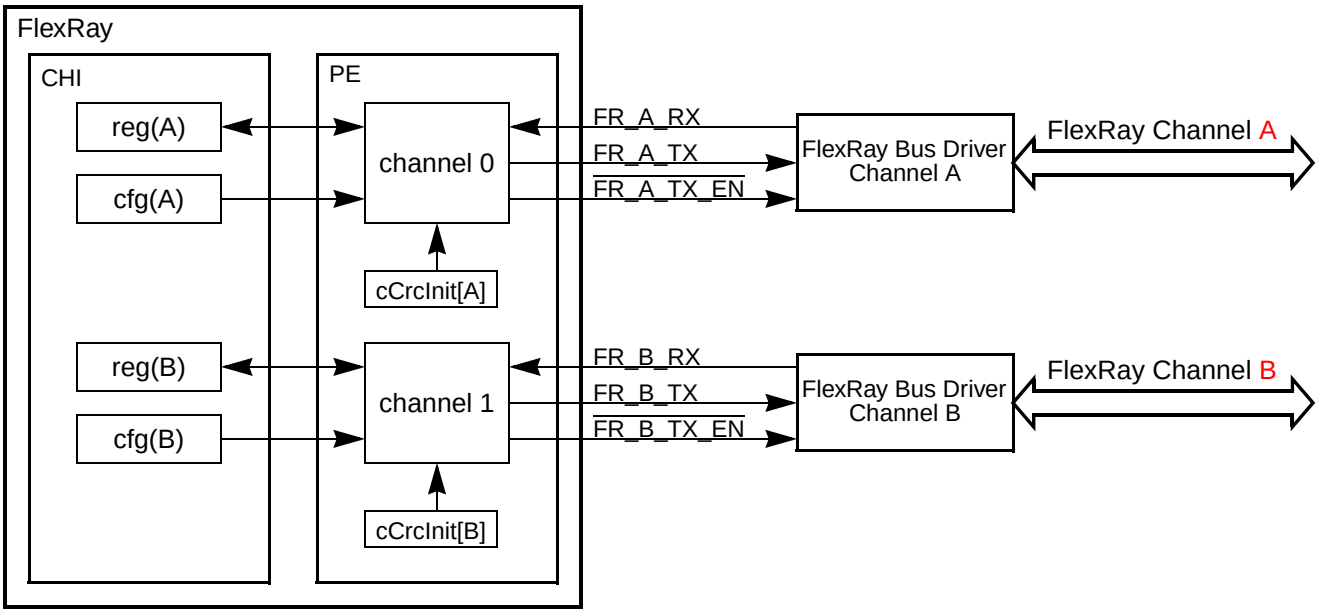


Figure 26-139. Dual Channel Device Mode

26.6.10.2 Single Channel Device Mode

The single channel device mode supports devices that have only one FlexRay port available. This FlexRay port consists of the signals FR_A_RX, FR_A_TX, and $\overline{\text{FR\_A\_TX\_EN}}$ and can be connected to either the physical bus channel A (shown in Figure 26-140) or the physical bus channel B (shown in Figure 26-141).

If the device is configured as a single channel device by setting MCR.SCD to 1, only the internal channel A and the FlexRay Port A is used. Depending on the setting of MCR.CHA and MCR.CHB, the internal channel A behaves either as a FlexRay Channel A or FlexRay Channel B. The bit MCR.CHA must be set, if the FlexRay Port A is connected to a FlexRay Channel A. The bit MCR.CHB must be set if the FlexRay Port A is connected to a FlexRay Channel B. The two FlexRay channels differ only in the initial value for the frame CRC *cCrcInit*. For a single channel device, the application can access and configure only the registers related to internal channel A.

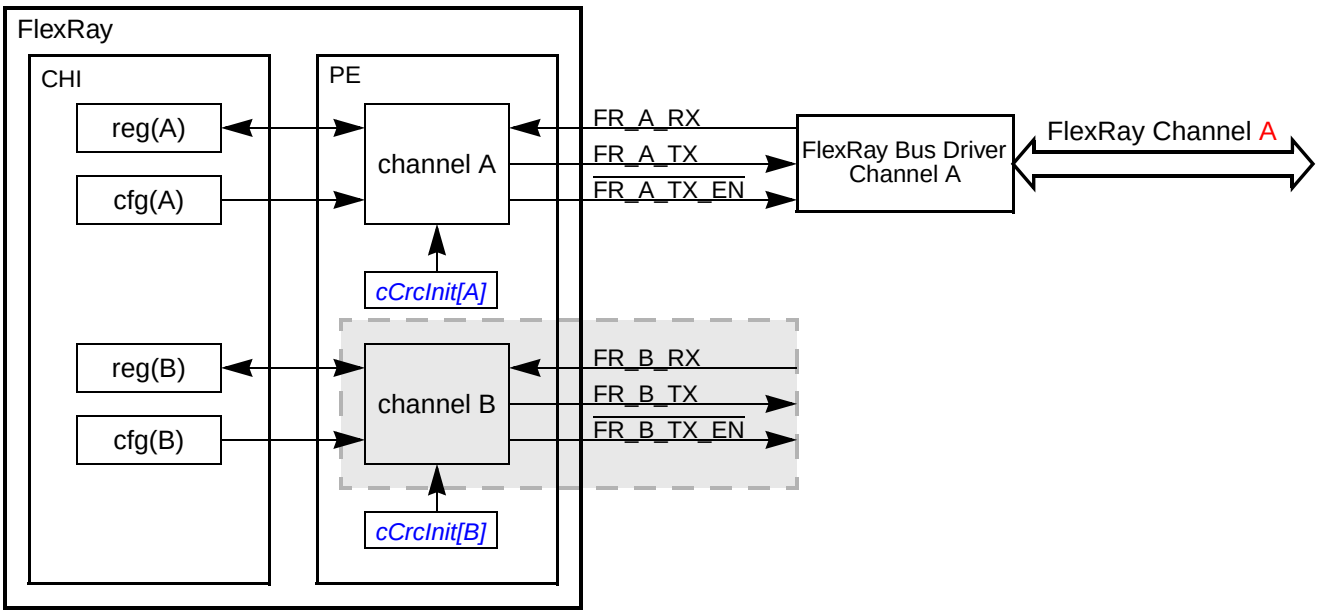


Figure 26-140. Single Channel Device Mode (Channel A)

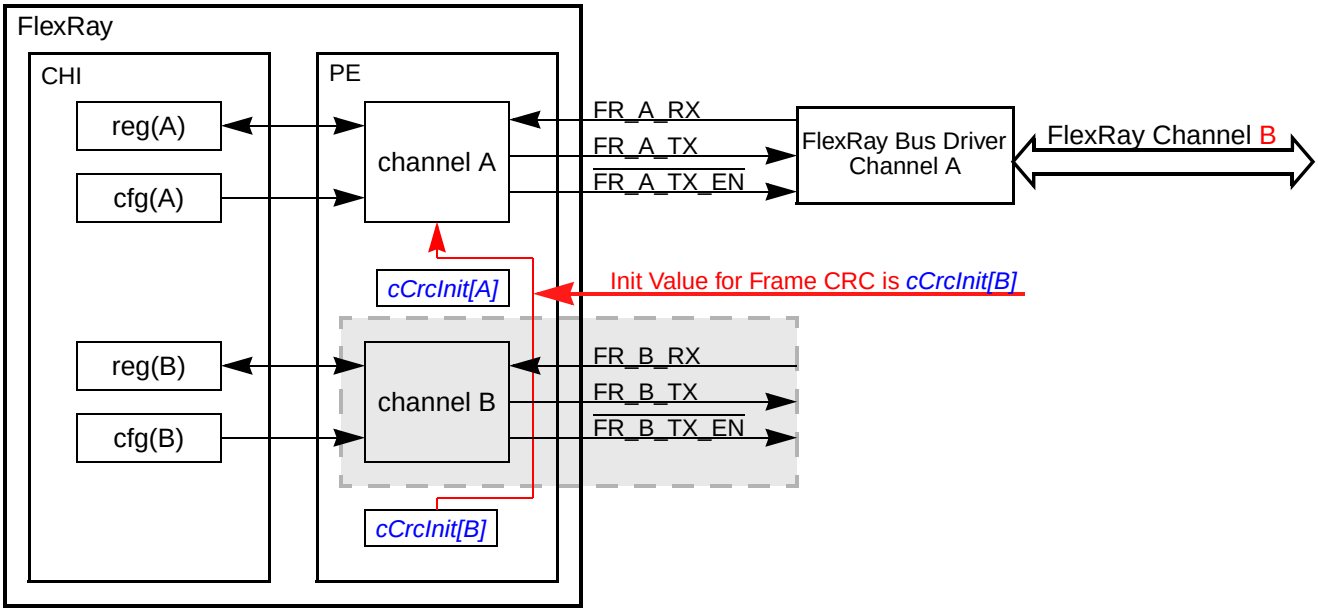


Figure 26-141. Single Channel Device Mode (Channel B)

26.6.11 External Clock Synchronization

The application of the external rate and offset correction is triggered when the application writes to the EOC_AP and ERC_AP fields in the Protocol Operation Control Register (POCR). The PE applies the external correction values in the next even-odd cycle pair as shown in Figure 26-142 and Figure 26-143.

NOTE

The values provided in the EOC_AP and ERC_AP fields are the values that were written from the application most recently. If these value were already applied, they will not be applied in the current cycle pair again.

If the offset correction applied in the NIT of cycle 2n+1 shall be affect by the external offset correction, the EOC_AP field must be written to after the start of cycle 2n and before the end of the static segment of cycle 2n+1. If this field is written to after the end of the static segment of cycle 2n+1, it is not guaranteed that the external correction value is applied in cycle 2n+1. If the value is not applied in cycle 2n+1, then the value will be applied in the cycle 2n+3. Refer to Figure 26-142 for timing details.

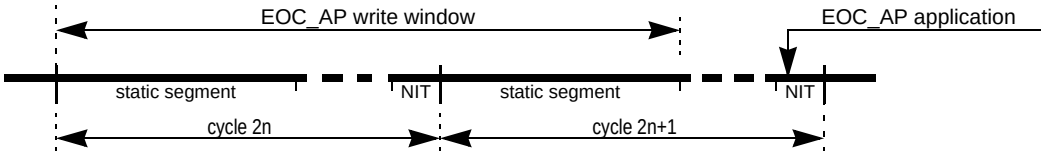


Figure 26-142. External Offset Correction Write and Application Timing

If the rate correction for the cycle pair [2n+2, 2n+3] shall be affect by the external offset correction, the ERC_AP field must be written to after the start of cycle 2n and before the end of the static segment start of cycle 2n+1. If this field is written to after the end of the static segment of cycle 2n+1, it is not guaranteed that the external correction value is applied in cycle pair [2n+2, 2n+3]. If the value is not applied for cycle pair [2n+2, 2n+3], then the value will be applied for cycle pair [2n+4, 2n+5]. Refer to Figure 26-143 for details.

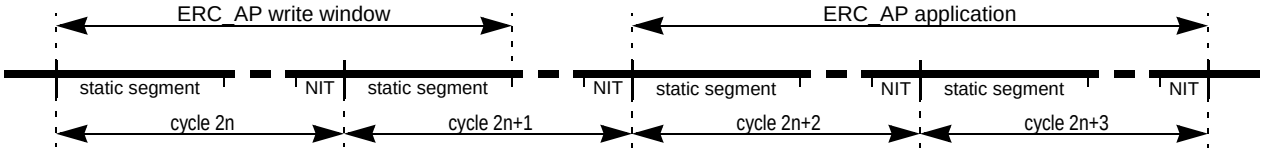


Figure 26-143. External Rate Correction Write and Application Timing

26.6.12 Sync Frame ID and Sync Frame Deviation Tables

The FlexRay protocol requires the provision of a snapshot of the Synchronization Frame ID tables for the even and odd communication cycle for both channels. The controller provides the means to write a copy of these internal tables into the FlexRay memory and ensures application access to consistent tables by means of table locking. Once the application has locked the table successfully, the controller will not overwrite these tables and the application can read a consistent snapshot.

NOTE

Only synchronization frames that have passed the synchronization frame filters are considered for clock synchronization and appear in the sync frame tables.

26.6.12.1 Sync Frame ID Table Content

The Sync Frame ID Table is a snapshot of the protocol related variables *vsSyncIdListA* and *vsSyncIdListB* for each even and odd communication cycle. This table provides a list of the frame IDs of the synchronization frames received on the corresponding channel and cycle that are used for the clock synchronization.

26.6.12.2 Sync Frame Deviation Table Content

The Sync Frame Deviation Table is a snapshot of the protocol related variable zsDev(id)(oe)(ch)!Value. Each Sync Frame Deviation Table entry provides the deviation value for the sync frame, with the frame ID presented in the corresponding entry in the Sync Frame ID Table.

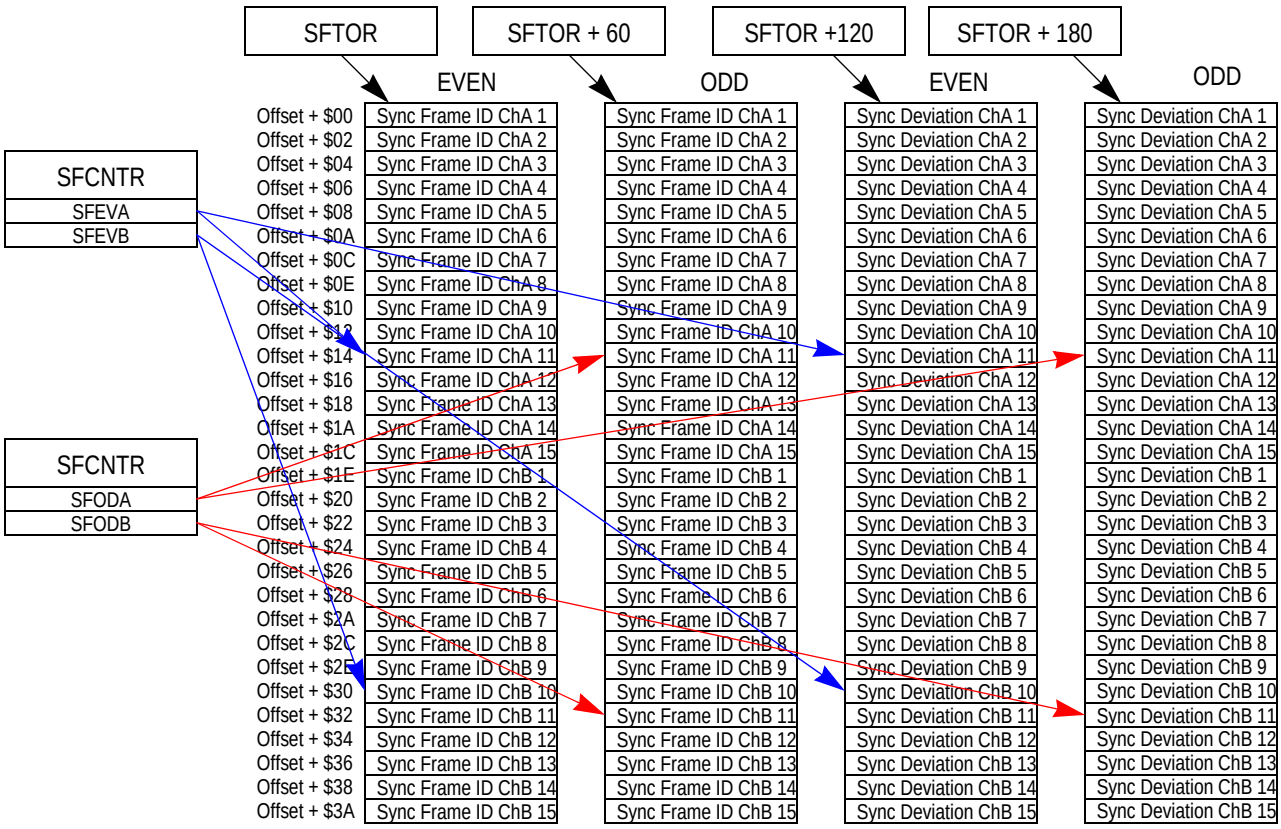


Figure 26-144. Sync Table Memory Layout

26.6.12.3 Sync Frame ID and Sync Frame Deviation Table Setup

The controller writes a copy of the internal synchronization frame ID and deviation tables into the FlexRay memory if requested by the application. The application must provide the appropriate amount of FlexRay memory for the tables. The memory layout of the tables is given in Figure 26-144. Each table occupies 120 16-bit entries.

While the protocol is in *POC:config* state, the application must program the offsets for the tables into the Sync Frame Table Offset Register (SFTOR).

26.6.12.4 Sync Frame ID and Sync Frame Deviation Table Generation

The application controls the generation process of the Sync Frame ID and Sync Frame Deviation Tables into the FlexRay memory using the Sync Frame Table Configuration, Control, Status Register (SFTCCSR). A summary of the copy modes is given in Table 26-114.

Table 26-114. Sync Frame Table Generation Modes

SFTCCSR			Description
OPT	SDVEN	SIDEN	
0	0	0	No Sync Frame Table copy
0	0	1	Sync Frame ID Tables will be copied continuously
0	1	0	Reserved
0	1	1	Sync Frame ID Tables and Sync Frame Deviation Tables will be copied continuously
1	0	0	No Sync Frame Table copy
1	0	1	Sync Frame ID Tables for next even-odd-cycle pair will be copied
0	1	0	Reserved
1	1	1	Sync Frame ID Tables and Sync Frame Deviation Tables for next even-odd-cycle pair will be copied

The Sync Frame Table generation process is described in the following for the even cycle. The same sequence applies to the odd cycle.

If the application has enabled the sync frame table generation by setting SFTCCSR.SIDEN to 1, the controller starts the update of the even cycle related tables after the start of the NIT of the next even cycle. The controller checks if the application has locked the tables by reading the SFTCCSR.ELKS lock status bit. If this bit is set, the controller will not update the table in this cycle. If this bit is cleared, the controller locks this table and starts the table update. To indicate that these tables are currently updated and may contain inconsistent data, the controller clears the even table valid status bit SFTCCSR[EVAL]. Once all table entries related to the even cycle have been transferred into the FlexRay memory, the controller sets the even table valid bit SFTCCSR[EVAL] and the Even Cycle Table Written Interrupt Flag EVT_IF in the Protocol Interrupt Flag Register 1 (PIFR1). If the interrupt enable flag EVT_IE is set, an interrupt request is generated.

To read the generated tables, the application must lock the tables to prevent the controller from updating these tables. The locking is initiated by writing a 1 to the even table lock trigger SFTCCSR.ELKT. When the even table is not currently updated by the controller, the lock is granted and the even table lock status bit SFTCCSR.ELKS is set. This indicates that the application has successfully locked the even sync tables and the corresponding status information fields SFRA, SFRB in the Sync Frame Counter Register (SFCNTR). The value in the SFTCCSR.CYCNUM field provides the number of the cycle that this table is related to.

The number of available table entries per channel is provided in the SFCNTR.SFEVA and SFCNTR.SFEVB fields. The application can now start to read the sync table data from the locations given in Figure 26-144.

After reading all the data from the locked tables, the application must unlock the table by writing to the even table lock trigger SFTCCSR.ELKT again. The even table lock status bit SFTCCSR.ELKS is reset immediately.

If the sync frame table generation is disabled, the table valid bits SFTCCSR[EVAL] and SFTCCSR[EVAL] are reset when the counter values in the Sync Frame Counter Register (SFCNTR) are updated. This is done because the tables stored in the FlexRay memory are no longer related to the values in the Sync Frame Counter Register (SFCNTR).

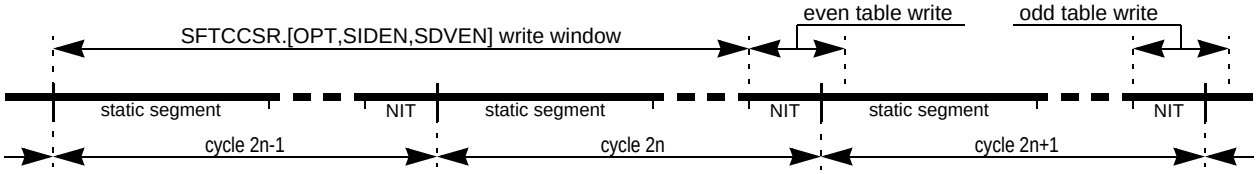


Figure 26-145. Sync Frame Table Trigger and Generation Timing

26.6.12.5 Sync Frame Table Access

The sync frame tables will be transferred into the FlexRay memory during the table write windows shown in Figure 26-145. During the table write, the application cannot lock the table that is currently written. If the application locks the table outside of the table write window, the lock is granted immediately.

26.6.12.5.1 Sync Frame Table Locking and Unlocking

The application locks the even/odd sync frame table by writing 1 to the lock trigger bit ELKT/OLKT in the Sync Frame Table Configuration, Control, Status Register (SFTCCSR). If the affected table is not currently written to the FlexRay memory, the lock is granted immediately, and the lock status bit ELKS/OLKS is set. If the affected table is currently written to the FlexRay memory, the lock is not granted. In this case, the application must issue the lock request again until the lock is granted.

The application unlocks the even/odd sync frame table by writing 1 to the lock trigger bit ELKT/OLKT. The lock status bit ELKS/OLKS is cleared immediately.

26.6.13 MTS Generation

The controller provides a flexible means to request the transmission of the Media Access Test Symbol MTS in the symbol window on channel A or channel B.

The application can configure the set of communication cycles in which the MTS will be transmitted over the FlexRay bus by programming the CYCCNTMSK and CYCCNTVAL fields in the MTS A Configuration Register (MTSACFR) and MTS B Configuration Register (MTSBCFR).

The application enables or disables the generation of the MTS on either channel by setting or clearing the MTE control bit in the MTS A Configuration Register (MTSACFR) or MTS B Configuration Register (MTSBCFR). If an MTS is to be transmitted in a certain communication cycle, the application must set the MTE control bit during the static segment of the preceding communication cycle.

The MTS is transmitted over channel A in the communication cycle with number CYCCNT, if Equation 26-20, Equation 26-21, and Equation 26-21 are fulfilled.

PSR0[PROTSTATE] = *POC:normal active* **Eqn. 26-19**

MTSACRF[MTE] = 1 **Eqn. 26-20**

YCCNT & MTSACFR[CYCCNTMSK] = MTSACFR[CYCCNTVAL] & MTSACFR[CYCCNTMSK] **Eqn. 26-21**

The MTS is transmitted over channel B in the communication cycle with number CYCCNT, if Equation 26-19, Equation 26-22, and Equation 26-23 are fulfilled.

MTSBCRF[MTE] = 1 **Eqn. 26-22**

YCCNT & MTSBCFR[CYCCNTMSK] = MTSBCFR[CYCCNTVAL] & MTSBCFR[CYCCNTMSK] **Eqn. 26-23**

26.6.14 Key Slot Transmission

26.6.14.1 Key Slot Assignment

A key slot is assigned to the controller if the key_slot_id field in the Protocol Configuration Register 18 (PCR18) is configured with a value greater than 0 and less or equal to number_of_static_slots in Protocol Configuration Register 2 (PCR2), otherwise no key slot is assigned.

26.6.14.2 Key Slot Transmission in *POC:startup*

If a key slot is assigned and the controller is in the *POC:startup* state, startup null frames will be transmitted as specified by FlexRay Communications System Protocol Specification, Version 2.1 Rev A.

26.6.14.3 Key Slot Transmission in *POC:normal active*

If a key slot is assigned and the controller is in *POC:normal active*, a frame of the type as shown in Table 26-115 is transmitted. If a transmit message buffer is configured for the key slot and a valid message is available, a message frame is transmitted (see Section 26.6.6.2.5, Message Transmission). If no transmit message buffer is configured for the key slot or no valid message is available, a null frame is transmitted (see Section 26.6.6.2.6, Null Frame Transmission).

Table 26-115. Key Slot Frame Type

PCR11[key_slot_used_for_sync]	PCR11[key_slot_used_for_startup]	key slot frame type
0	0	normal frame
0	1	normal frame[1]
1	0	sync frame
1	1	startup frame

[1] The frame transmitted has an semantically incorrect header and will be detected as an invalid frame at the receiver.

26.6.15 Sync Frame Filtering

Each received synchronization frame must pass the Sync Frame Acceptance Filter and the Sync Frame Rejection Filter before it is considered for clock synchronization. If the synchronization frame filtering is

globally disabled (the SFFE control bit in the Module Configuration Register (MCR) is cleared), all received synchronization frames are considered for clock synchronization. If a received synchronization frame did not pass at least one of the two filters, this frame is processed as a normal frame and is not considered for clock synchronization.

26.6.15.1 Sync Frame Acceptance Filtering

The synchronization frame acceptance filter is implemented as a value-mask filter. The value is configured in the Sync Frame ID Acceptance Filter Value Register (SFIDAFVR) and the mask is configured in the Sync Frame ID Acceptance Filter Mask Register (SFIDAFMR). A received synchronization frame with the frame ID FID passes the sync frame acceptance filter, if Equation 26-24 or Equation 26-25evaluates to true.

$$MCR[SFFE] = 0 \qquad \textit{Eqn. 26-24}$$

$$ID \,\&\, SFIDAFMR[FMSK] = SFIDAFVR[FVAL] \,\&\, SFIDAFMR[FMSK \qquad \textit{Eqn. 26-25}$$

NOTE

Sync frames are transmitted in the static segment only. Thus FID $\leq$ 1023.

26.6.15.2 Sync Frame Rejection Filtering

The synchronization frame rejection filter is a comparator. The compare value is defined by the Sync Frame ID Rejection Filter Register (SFIDRFR). A received synchronization frame with the frame ID FID passes the sync frame rejection filter if Equation 26-26 or Equation 26-27 evaluates to true.

$$MCR[SFFE] = 0 \qquad \textit{Eqn. 26-26}$$

$$FID \neq SFIDRFR[SYNFRID] \qquad \textit{Eqn. 26-27}$$

NOTE

Sync frames are transmitted in the static segment only. Thus FID $\leq$ 1023.

26.6.16 Strobe Signal Support

The controller provides a number of strobe signals for observing internal protocol timing related signals in the protocol engine. The signals are listed and described in Table 26-12.

26.6.16.1 Strobe Signal Assignment

Each of the strobe signals listed in Table 26-12 can be assigned to one of the four strobe ports using the Strobe Signal Control Register (STBSCR). To assign multiple strobe signals, the application must write multiple times to the Strobe Signal Control Register (STBSCR) with appropriate settings.

To read out the current settings for a strobe signal with number N, the application must execute the following sequence.

1. Write to STBSCR with WMD = 1 and SEL = N. (updates SEL field only)

2. Read STBCSR.
   The SEL field provides N and the ENB and STBPSEL fields provides the settings for signal N.

26.6.16.2 Strobe Signal Timing

This section provides detailed timing information of the strobe signals with respect to the protocol engine clock.

The strobe signals display internal PE signals. Due to the internal architecture of the PE, some signals are generated several PE clock cycles before the actual action is performed on the FlexRay Bus. These signals are listed in Table 26-12 with a negative clock offset. An example waveform is given in Figure 26-146.

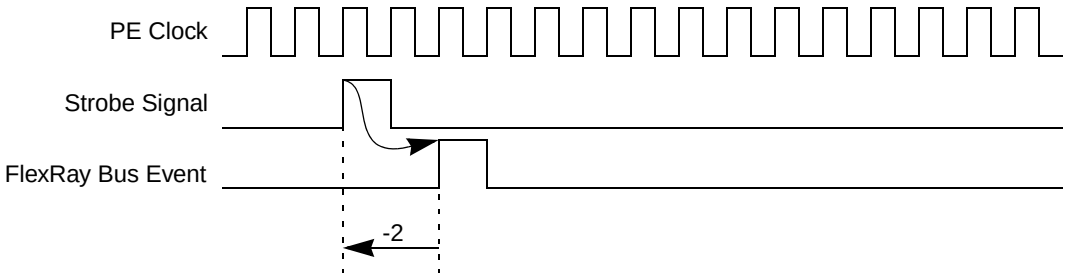


Figure 26-146. Strobe Signal Timing (type = pulse, clk_offset = -2)

Other signals refer to events that occurred on the FlexRay Bus some cycles before the strobe signal is changed. These signals are listed in Table 26-12 with a positive clock offset. An example waveform is given in Figure 26-147.

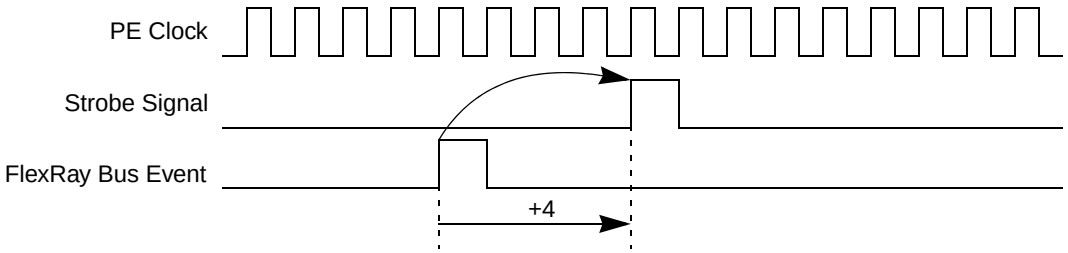


Figure 26-147. Strobe Signal Timing (type = pulse, clk_offset = +4)

26.6.17 Timer Support

The controller provides two timers, which run on the FlexRay time base. Each timer generates a maskable interrupt when it reaches a configured point in time. Timer T1 is an absolute timer. Timer T2 can be configured to be an absolute or a relative timer. Both timers can be configured to be repetitive. In the non-repetitive mode, timer stops if it expires. In repetitive mode, timer is restarted when it expires.

Both timers are active only when the protocol is in *POC:normal active* or *POC:normal passive* state. If the protocol is not in one of these modes, the timers are stopped. The application must restart the timers when the protocol has reached the *POC:normal active* or *POC:normal passive* state.

26.6.17.1 Absolute Timer T1

The absolute timer T1 has the protocol cycle count and the macrotick count as the time base. The timer 1 interrupt flag TI1_IF in the Protocol Interrupt Flag Register 0 (PIFR0) is set at the macrotick start event, if Equation 26-28 and Equation 26-29 are fulfilled.

Eqn. 26-28

$$CYCTR[CTCCNT] \,\&\, TI1CYSR[T1\_CYC\_MSK] = TI1CYSR[T1\_CYC\_VAL] \,\&\, TI1CYSR[T1\_CYC\_MSK]$$

$$MTCTR[MTCT] = TI1MTOR[T1\_MTOFFSET] \qquad \textbf{\textit{Eqn. 26-29}}$$

If the timer 1 interrupt enable bit TI1_IE in the Protocol Interrupt Enable Register 0 (PIER0) is asserted, an interrupt request is generated.

The status bit T1ST is set when the timer is triggered, and is cleared when the timer expires and is non-repetitive. If the timer expires but is repetitive, the T1ST bit is not cleared and the timer is restarted immediately. The T1ST is cleared when the timer is stopped.

26.6.17.2 Absolute / Relative Timer T2

The timer T2 can be configured to be an absolute or relative timer by setting the T2_CFG control bit in the Timer Configuration and Control Register (TICCR). The status bit T2ST is set when the timer is triggered, and is cleared when the timer expires and is non-repetitive. If the timer expires but is repetitive, the T2ST bit is not cleared and the timer is restarted immediately. The T2ST is cleared when the timer is stopped.

26.6.17.2.1 Absolute Timer T2

If timer T2 is configured as an absolute timer, it has the same functionality timer T1 but the configuration from Timer 2 Configuration Register 0 (TI2CR0) and Timer 2 Configuration Register 1 (TI2CR1) is used. On expiration of timer T2, the interrupt flag TI2_IF in the Protocol Interrupt Flag Register 0 (PIFR0) is set. If the timer 1 interrupt enable bit TI1_IE in the Protocol Interrupt Enable Register 0 (PIER0) is asserted, an interrupt request is generated.

26.6.17.2.2 Relative Timer T2

If the timer T2 is configured as a relative timer, the interrupt flag TI2_IF in the Protocol Interrupt Flag Register 0 (PIFR0) is set, when the programmed amount of macroticks MT[31:0], defined by Timer 2 Configuration Register 0 (TI2CR0) and Timer 2 Configuration Register 1 (TI2CR1), has expired since the trigger or restart of timer 2. The relative timer is implemented as a down counter and expires when it has reached 0. At the macrotick start event, the value of MT[31:0] is checked and then decremented. Thus, if the timer is started with MT[31:0] == 0, it expires at the next macrotick start.

26.6.18 Slot Status Monitoring

The controller provides several means for slot status monitoring. All slot status monitors use the same slot status vector provided by the PE. The PE provides a slot status vector for each static slot, for each dynamic slot, for the symbol window, and for the NIT, on a per channel base. The content of the slot status vector

is described in Table 26-116. The PE provides the slot status vector within the first macrotick after the end of the related slot/window/NIT, as shown in Figure 26-148.

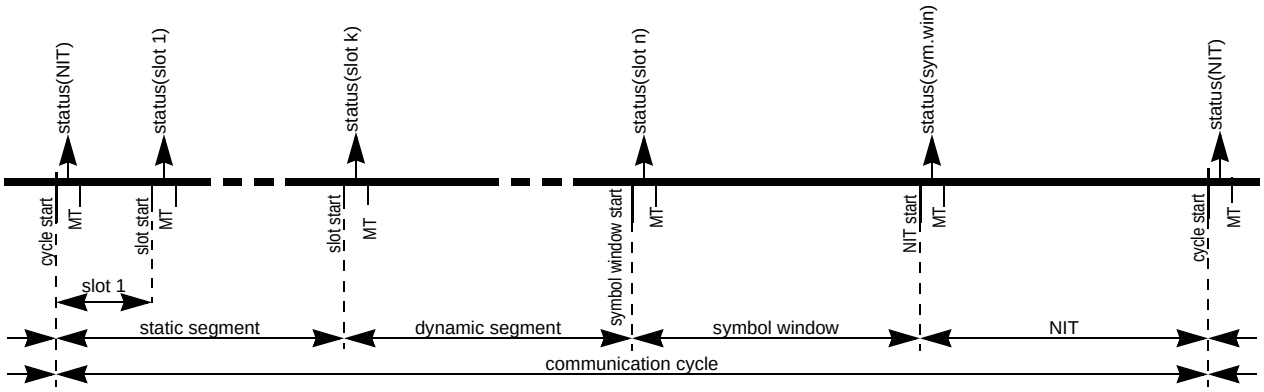


Figure 26-148. Slot Status Vector Update

NOTE

The slot status for the NIT of cycle *n* is provided after the start of cycle *n* + 1.

Table 26-116. Slot Status Content

	Status Content

static / dynamic Slot	slot related status
	vSS!ValidFrame – valid frame received
	vSS!SyntaxError – syntax error occurred while receiving
	vSS!ContentError– content error occurred while receiving
	vSS!BViolation – boundary violation while receiving
	for slots in which the module transmits:
	vSS!TxConflict – reception ongoing while transmission starts
	for slots in which the module does not transmit:
	vSS!TxConflict – reception ongoing while transmission starts
	first valid – channel that has received the first valid frame
	received frame related status
	extracted from
	a) header of valid frame, if vSS!ValidFrame = 1
	b) last received header, if vSS!ValidFrame = 0
	c) set to 0, if nothing was received
	vRF!Header!NFIndicator – Null Frame Indicator (0 for null frame)
	vRF!Header!SuFIndicator – Startup Frame Indicator
	vRF!Header!SyFIndicator – Sync Frame Indicator

Symbol Window	window related status *vSS!ValidFrame* – always 0 *vSS!ContentError* – content error occurred while receiving *vSS!SyntaxError* – syntax error occurred while receiving *vSS!BViolation* – boundary violation while receiving *vSS!TxConflict* – reception ongoing while transmission starts received symbol related status *vSS!ValidMTS* – valid Media Test Access Symbol received received frame related status see static/dynamic slot
NIT	NIT related status *vSS!ValidFrame* – always 0 *vSS!ContentError* – content error occurred while receiving *vSS!SyntaxError* – syntax error occurred while receiving *vSS!BViolation* – boundary violation while receiving *vSS!TxConflict* – always 0 received frame related status see static/dynamic slot

26.6.18.1 Channel Status Error Counter Registers

The two channel status error counter registers, Channel A Status Error Counter Register (CASERCR) and Channel B Status Error Counter Register (CBSERCR), incremented by one, if at least one of four slot status error bits, *vSS!SyntaxError*, *vSS!ContentError*, *vSS!BViolation*, or *vSS!TxConflict* is set to 1. The status vectors for all slots in the static and dynamic segment, in the symbol window, and in the NIT are taken into account. The counters wrap round after they have reached the maximum value.

26.6.18.2 Protocol Status Registers

The Protocol Status Register 2 (PSR2) provides slot status information about the Network Idle Time NIT and the Symbol Window. The Protocol Status Register 3 (PSR3) provides aggregated slot status information.

26.6.18.3 Slot Status Registers

The eight slot status registers, Slot Status Registers (SSR0–SSR7), can be used to observe the status of static slots, dynamic slots, the symbol window, or the NIT without individual message buffers. These registers provide all slot status related and received frame / symbol related status information, as given in Table 26-116, except of the *first valid* indicator for non-transmission slots.

26.6.18.4 Slot Status Counter Registers

The controller provides four slot status error counter registers, Slot Status Counter Registers (SSCR0–SSCR3). Each of these slot status counter registers is updated with the value of an internal slot status counter at the start of a communication cycle. The internal slot status counter is incremented if its increment condition, defined by the Slot Status Counter Condition Register (SSCCR), matches the status vector provided by the PE. All static slots, the symbol window, and the NIT status are taken into account. *Dynamic* slots are *excluded*. The internal slot status counting and update timing is shown in Figure 26-149.

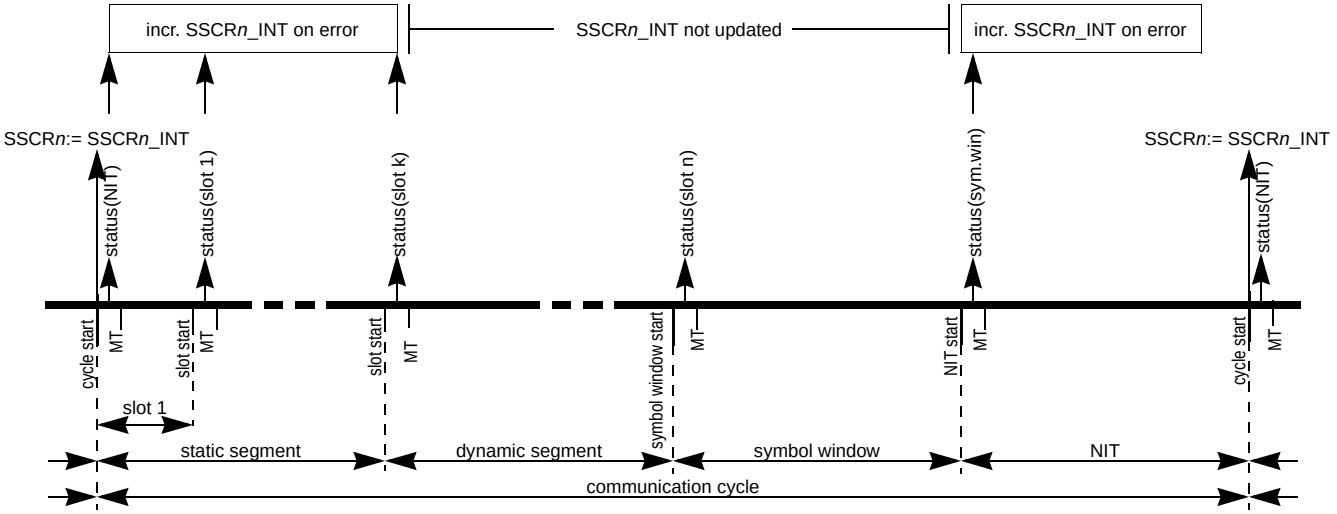


Figure 26-149. Slot Status Counting and SSCR*n* Update

The PE provides the status of the NIT in the first slot of the next cycle. Due to these facts, the SSCR*n* register reflects, in cycle n, the status of the NIT of cycle $n - 2$, and the status of all static slots and the symbol window of cycle $n - 1$.

The increment condition for each slot status counter consists of two parts, the frame related condition part and the slot related condition part. The internal slot status counter SSCR*n*_INT is incremented if at least one of the conditions is fulfilled:

1. frame related condition:

- (SSCCR*n*.VFR | SSCCR*n*.SYF | SSCCR*n*.NUF | SSCCR*n*.SUF) // count on frame condition = 1;

and

- ((~SSCCR*n*.VFR | *vSS!ValidFrame*) & // valid frame restriction
  (~SSCCR*n*.SYF | *vRF!Header!SyFIndicator*) & // sync frame indicator restriction
  (~SSCCR*n*.NUF | ~*vRF!Header!NFIndicator*) & // null frame indicator restriction
  (~SSCCR*n*.SUF | *vRF!Header!SuFIndicator*)) // startup frame indicator restriction = 1;

NOTE

The indicator bits SYF, NUF, and SUF are valid only when a valid frame was received. Thus it is required to set the VFR always, whenever count on frame condition is used.

2. slot related condition:

- ((SSCCR*n*.STATUSMASK[3] & *vSS!ContentError*) | // increment on content error
  (SSCCR*n*.STATUSMASK[2] & *vSS!SyntaxError*) | // increment on syntax error
  (SSCCR*n*.STATUSMASK[1] & *vSS!BViolation*) | // increment on boundary violation
  (SSCCR*n*.STATUSMASK[0] & *vSS!TxConflict*)) // increment on transmission conflict = 1;

If the slot status counter is in single cycle mode (SSCCR*n*.MCY = 0), the internal slot status counter SSCR*n*_INT is reset at each cycle start. If the slot status counter is in the multicycle mode (SSCCR*n*.MCY = 1), the counter is not reset and incremented, until the maximum value is reached.

26.6.18.5 Message Buffer Slot Status Field

Each individual message buffer and each FIFO message buffer provides a slot status field, which provides the information shown in Table 26-116 for the static/dynamic slot. The update conditions for the slot status field depend on the message buffer type. Refer to the Message Buffer Update Sections in Section 26.6.6, Individual Message Buffer Functional Description.

26.6.19 System Bus Access

This section provides a description of the system bus accesses performed by the controller.

All FlexRay memory data located in the system memory are accessed via the system bus. There are two types of failures that can occur during the system bus access, the system bus illegal address access and the system bus access timeout.

The behavior of the controller after the occurrence of a system bus failure is defined by the SBFF bit in the Module Configuration Register (MCR).

26.6.19.1 System Bus Illegal Address Access

If the system bus detects an controller access to an illegal address, the controller receives a notification from the system bus about this event and sets the ILSA_EF flag in the CHI Error Flag Register (CHIERFR).

26.6.19.2 System Bus Access Timeout

The controller starts a timer when it has send an access request to the system bus. This timer expires after 2 * SYMATOR.TIMEOUT + 2 system bus clock cycles. If the access is not finished within this amount of time, the SBCF_EF flag in the CHI Error Flag Register (CHIERFR) is set.

NOTE

If the system memory read access that retrieves the first message buffer header data from a FlexRay transmit buffer fails due to a system memory access timeout or illegal address access, it is possible that the slot status information for the previous slot is written into the currently used transmit message buffer. In this case, the slot status information is not written into the message buffer assigned to the last slot. Thus, both the message buffer assigned to the last slot, and the currently used transmit message buffer contain incorrect slot status information. However, if this occurs, either the System Bus Communication Failure Error Flag (SBCF_EF) or the Illegal System Bus Address Error Flag (ILSA_EF) will be set in the Controller Host Interface Error Flag Register (CHIERFR).

The FlexRay module and the system memory subsystem should be configured to avoid the occurrence of system memory access timeouts and illegal address accesses. In case that one of the error flags CHIERFR[SBCF_EF] or CHIERFR[ILSA_EF] is set, the application should not use the slot status information of the message buffers.

26.6.19.3 Continue after System Bus Failure

If the SBFF bit in the Module Configuration Register (MCR) is 0, the controller will continue its operation after the occurrence of the system bus access failure but will not generate any system bus accesses until the start of the next communication cycle.

If a frame is under transmission when the system bus failure occurs, a correct frame is generated with the remaining header and frame data are replaced by all zeros. Depending on the point in time this can affect the PPI bit, the Header CRC, the Payload Length in case of an dynamic slot, and the payload data. Starting from the next slot in the current cycle, no frames will be transmitted and received, except for the key slot, where a sync or startup null-frame is transmitted, if the key slot is assigned.

If a frame is received when the system bus failure occurs, the reception is aborted and the related receive message buffer is not updated.

Normal operation is resumed after the start of next communication cycle.

26.6.19.4 Freeze after System Bus Failure

If the SBFF bit in the Module Configuration Register (MCR) is set to 1, the controller will go into the freeze mode immediately after the occurrence of one of the system bus access failures.

26.6.20 Interrupt Support

The controller provides 172 individual interrupt sources and five combined interrupt sources.

26.6.20.1 Individual Interrupt Sources

26.6.20.1.1 Message Buffer Interrupts

The controller provides 128 message buffer interrupt sources.

Each individual message buffer provides an interrupt flag MBCCSR*n*[MBIF] and an interrupt enable bit MBCCSR*n*[MBIE]. The controller sets the interrupt flag when the slot status of the message buffer was updated. If the interrupt enable bit is asserted, an interrupt request is generated.

26.6.20.1.2 FIFO Interrupts

The controller provides 2 FIFO interrupt sources.

Each of the 2 FIFO provides a Receive FIFO Almost Full Interrupt Flag. The controller sets the Receive FIFO Almost Full Interrupt Flags (GIFER.FAFBIF, GIFER.FAFAIF) in the Global Interrupt Flag and Enable Register (GIFER) if the corresponding Receive FIFO fill level exceeds the defined watermark.

26.6.20.1.3 Wakeup Interrupt

The controller provides one interrupt source related to the wakeup.

The controller sets the Wakeup Interrupt Flag GIFER.WUPIF when it has received a wakeup symbol on the FlexRay bus. The controller generates an interrupt request if the interrupt enable bit GIFER.WUPIE is asserted.

26.6.20.1.4 Protocol Interrupts

The controller provides 25 interrupt sources for protocol related events. For details, see Protocol Interrupt Flag Register 0 (PIFR0) and Protocol Interrupt Flag Register 1 (PIFR1). Each interrupt source has its own interrupt enable bit.

26.6.20.1.5 CHI Error Interrupts

The controller provides 16 interrupt sources for CHI related error events. For details, see CHI Error Flag Register (CHIERFR). There is one common interrupt enable bit GIFER.CHIIE for all CHI error interrupt sources.

26.6.20.2 Combined Interrupt Sources

Each combined interrupt source generates an interrupt request only when at least one of the interrupt sources that is combined generates an interrupt request.

26.6.20.2.1 Receive Message Buffer Interrupt

The combined receive message buffer interrupt request RBIRQ is generated when at least one of the individual receive message buffers generates an interrupt request MBXIRQ[n] and the interrupt enable bit GIFER.RBIE is set.

26.6.20.2.2 Transmit Message Buffer Interrupt

The combined transmit message buffer interrupt request TBIRQ is generated when at least one of the individual transmit message buffers generates an interrupt request MBXIRQ[n] and the interrupt enable bit GIFER.TBIE is asserted.

26.6.20.2.3 Protocol Interrupt

The combined protocol interrupt request PRTIRQ is generated when at least one of the individual protocol interrupt sources generates an interrupt request and the interrupt enable bit GIFER.PRIE is set.

26.6.20.2.4 CHI Error Interrupt

The combined CHI error interrupt request CHIIRQ is generated when at least one of the individual chi error interrupt sources generates an interrupt request and the interrupt enable bit GIFER.CHIE is set.

26.6.20.2.5 Module Interrupt

The combined module interrupt request MIRQ is generated if at least one of the combined interrupt sources generates an interrupt request and the interrupt enable bit GIFER.MIE is set.

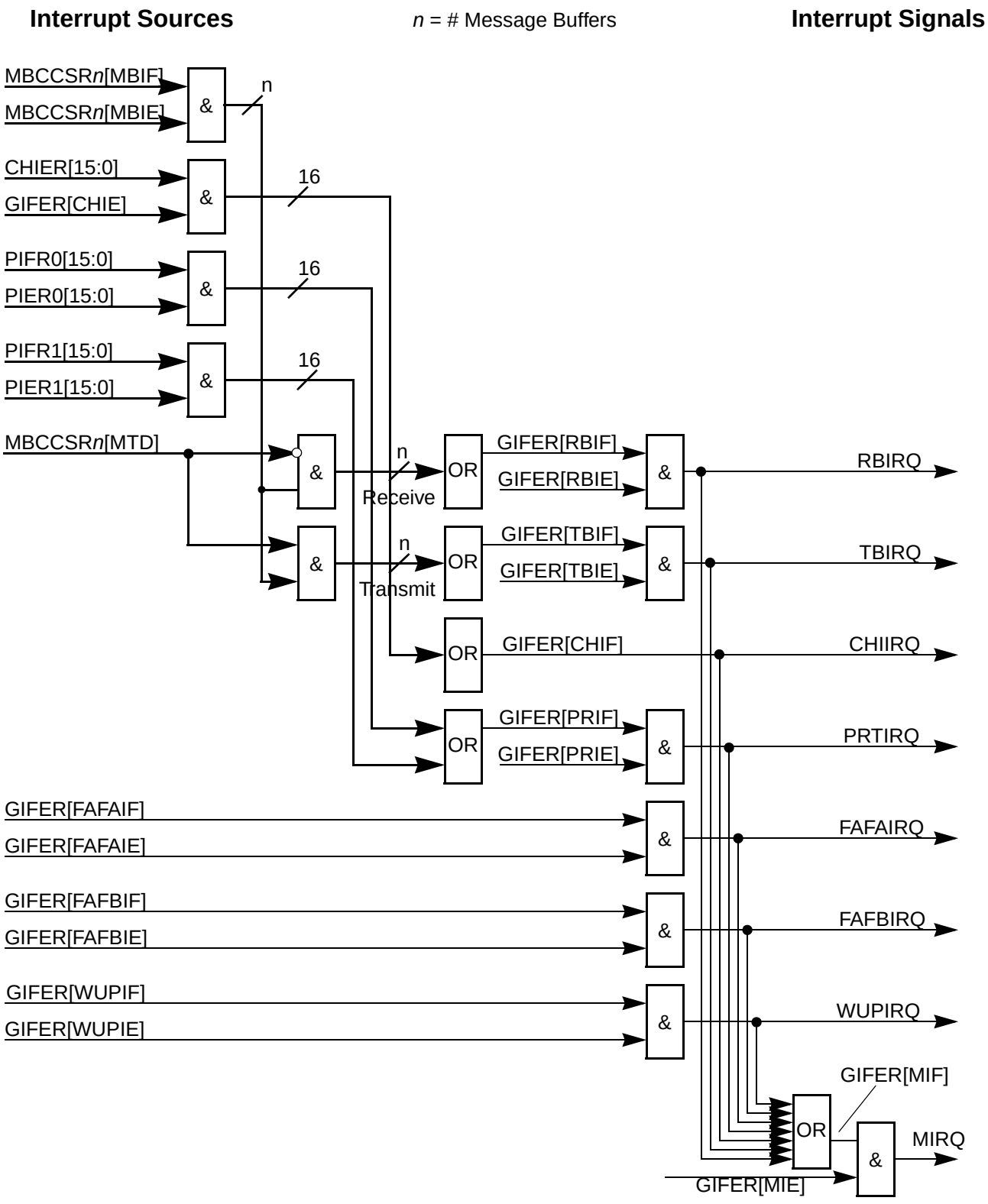


Figure 26-150. Scheme of cascaded interrupt request

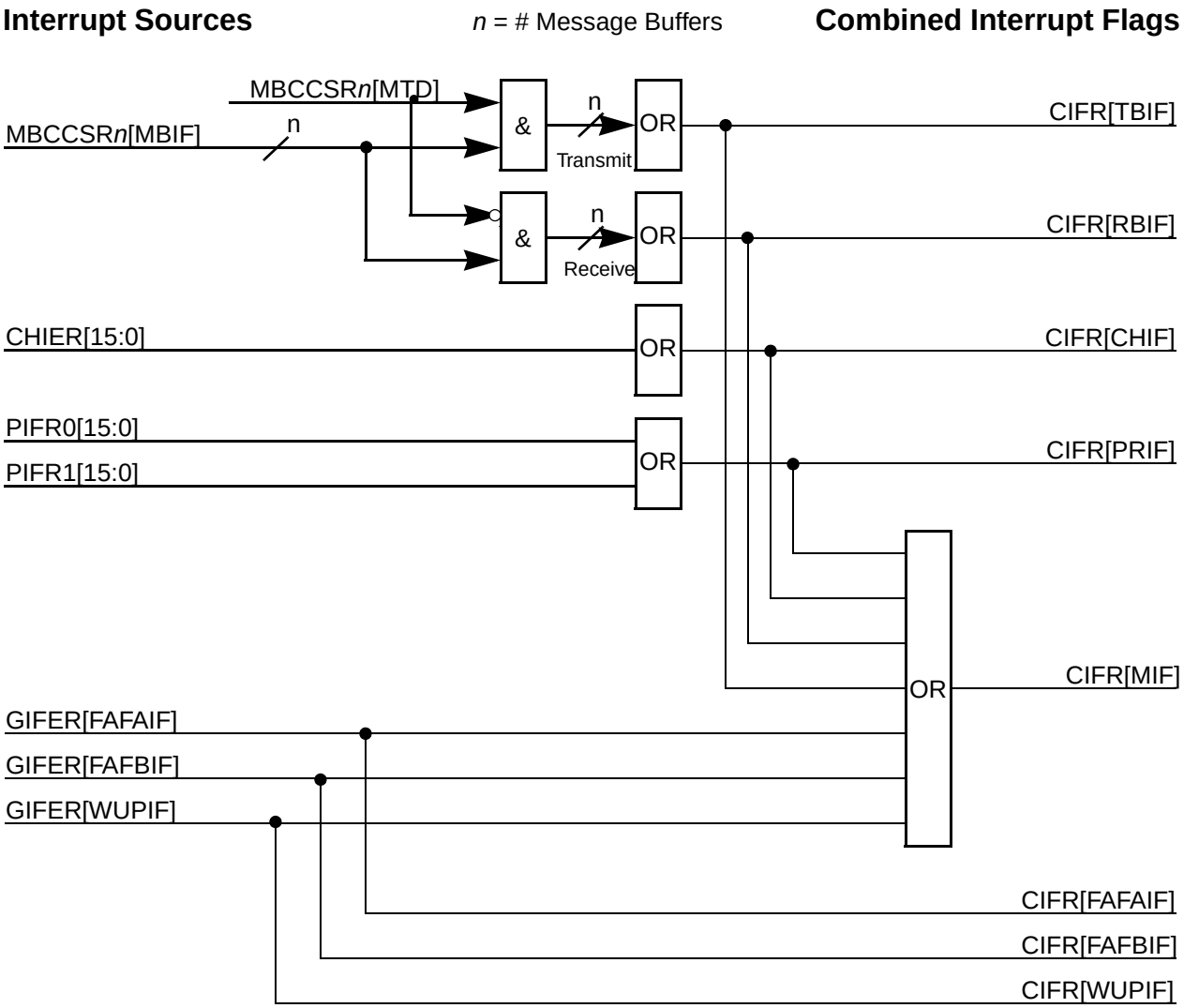


Figure 26-152. Scheme of combined interrupt flags

26.6.21 Lower Bit Rate Support

The controller supports a number of lower bit rates on the FlexRay bus channels. The lower bit rates are implemented by modifying the duration of the microtick *pdMicrotick*, the number of samples per microtick *pSamplesPerMicrotick*, the number of samples per bit *cSamplesPerBit*, and the strobe offset *cStrobeOffset*. The application configures the FlexRay channel bit rate by setting the BITRATE field in the Module Configuration Register (MCR). The protocol values are set internally. The available bit rates, the related BITRATE field configuration settings and related protocol parameter values are shown in Table 26-117.

Table 26-117. FlexRay Channel Bit Rate Control

FlexRay Channel Bit Rate [Mbit/s]	MCR.BITRATE	*pdMicrotick* [ns]	*gdSampleClockPeriod* [ns]	*pSamplesPerMicrotick*	*cSamplesPerBit*	*cStrobeOffset*
10.0	000	25.0	12.5	2	8	5
8.0	011	25.0	12.5	2	10	6
5.0	001	25.0	25.0	1	8	5
2.5	010	50.0	50.0	1	8	5

NOTE

The bit rate of 8 Mbit/s is not defined by the *FlexRay Communications System Protocol Specification, Version 2.1 Rev A.*

26.7 Application Information

26.7.1 Initialization Sequence

This section describes the required steps to initialize the controller. The first subsection describes the steps required after a system reset, the second section describes the steps required after preceding shutdown of the controller.

26.7.1.1 Module Initialization

This section describes the module related initialization steps after a system reset.

1. Configure controller.
   a) configure the control bits in the Module Configuration Register (MCR)
   b) configure the system memory base address in System Memory Base Address Register (SYMBADR)
2. Enable the controller.
   a) write 1 to the module enable bit MEN in the Module Configuration Register (MCR)

The controller now enters the Normal Mode. The application can commence with the protocol initialization described in Section 26.7.1.2, Protocol Initialization.

26.7.1.2 Protocol Initialization

This section describes the protocol related initialization steps.

1. Configure the Protocol Engine.
   a) issue CONFIG command via Protocol Operation Control Register (POCR)
   b) wait for *POC:config* in Protocol Status Register 0 (PSR0)
   c) configure the PCR0,..., PCR30 registers to set all protocol parameters
2. Configure the Message Buffers and FIFOs.
   a) set the number of message buffers used and the message buffer segmentation in the Message Buffer Segment Size and Utilization Register (MBSSUTR)
   b) define the message buffer data size in the Message Buffer Data Size Register (MBDSR)
   c) configure each message buffer by setting the configuration values in the Message Buffer Configuration, Control, Status Registers (MBCCSRn), Message Buffer Cycle Counter Filter Registers (MBCCFRn), Message Buffer Frame ID Registers (MBFIDRn), Message Buffer Index Registers (MBIDXRn)
   d) configure the FIFOs
   e) issue CONFIG_COMPLETE command via Protocol Operation Control Register (POCR)
   f) wait for *POC:ready* in Protocol Status Register 0 (PSR0)

After this sequence, the controller is configured as a FlexRay node and is ready to integrate into the FlexRay cluster.

26.7.2 Shut Down Sequence

This section describes a secure shut down sequence to stop the controller gracefully. The main targets of this sequence are

- finish all ongoing reception and transmission
- do not corrupt FlexRay bus and do not disturb ongoing FlexRay bus communication

For a graceful shutdown the application shall perform the following tasks:

1. Disable all enabled message buffers.
   a) repeatedly write 1 to MBCCSR*n*[EDT] until MBCCSR*n*[EDS] == 0.
2. Stop Protocol Engine.
   a) issue HALT command via Protocol Operation Control Register (POCR)
   b) wait for *POC:halt* in Protocol Status Register 0 (PSR0)

26.7.3 Number of Usable Message Buffers

This section describes the relationship between the number of message buffers that can be utilized and the required minimum CHI clock frequency. Additional constraints for the minimum CHI clock frequency are given in Section 26.3, Controller Host Interface Clocking.

The controller uses a sequential search algorithm to determine the individual message buffer assigned or subscribed to the next slot. This search must be finished within one FlexRay slot. The shortest FlexRay slot is an empty dynamic slot. An empty dynamic slot is a minislot and consists of *gdMinislot* macroticks with a nominal duration of *gdMacrotick*. The minimum duration of a corrected macrotick is $gdMacrotick_{min}$ = 39 µµT. This results in a minimum slot length of

$$\Delta_{slotmin} = 39 \cdot pdMicrotick \cdot gdMinislot \qquad \textbf{Eqn. 26-30}$$

The search engine is located in the CHI and runs on the CHI clock. It evaluates one individual message buffer per CHI clock cycle. For internal status update and double buffer commit operations, and as a result of the clock domain crossing jitter, an additional amount of 10 CHI clock cycles is required to ensure correct operation. For a given number of message buffers and for a given CHI clock frequency f_{chi}, this results in a search duration of

$$\Delta_{search} = \frac{1}{f_{chi}} \cdot (\#\ \text{MessageBuffers} + 10) \qquad \textbf{Eqn. 26-31}$$

The message buffer search must be finished within one slot which requires that Equation 26-32 must be fulfilled

$$\Delta_{search} \leq \Delta_{slotmin} \qquad \textbf{Eqn. 26-32}$$

This results in the formula given in Equation 26-33 which determines the required minimum CHI frequency for a given number of message buffers that are utilized.

$$f_{chi} \geq \frac{\#\ \text{MessageBuffers} + 10}{39 \cdot pdMicrotick \cdot gdMinislot} \qquad \textbf{Eqn. 26-33}$$

The minimum CHI frequency for a selected set of relevant protocol parameters is given in Table 26-118.

Table 26-118. Minimum f_{chi} [MHz] examples (128 message buffers)

pdMicrotick [ns]	*gdMinislot*					
	2	3	4	5	6	7
25.0	70.77	47.18	35.39	28.31	23.59	20.22
50.0	35.39	23.59	17.70	14.16	11.80	10.11

26.7.4 Protocol Control Command Execution

This section considers the issues of the protocol control command execution.

The application issues any of the protocol control commands listed in the POCCMD field of Table 26-15 by writing the command to the POCCMD field of the Protocol Operation Control Register (POCR). As a result the controller sets the BSY bit while the command is transferred to the PE. When the PE has accepted the command, the BSY flag is cleared. All commands are accepted by the PE.

The PE maintains a protocol command vector. For each command that was accepted by the PE, the PE sets the corresponding command bit in the protocol command vector. If a command is issued while the corresponding command bit is set, the command is not queued and is lost.

If the command execution block of the PE is idle, it selects the next accepted protocol command with the highest priority from the current protocol command vector according to the protocol control command priorities given in Table 26-119. If the current protocol state does not allow the execution of this protocol command (see POC state changes in *FlexRay Communications System Protocol Specification, Version 2.1 Rev A*) the controller asserts the illegal protocol command interrupt flag IPC_IF in the Protocol Interrupt Flag Register 1 (PIFR1). The protocol command is not executed in this case.

Some protocol commands may be interrupted by other commands or the detection of a fatal protocol error as indicated by Table 26-119. If the application issues the FREEZE or READY command, or if the PE detects a fatal protocol error, some commands already stored in the command vector will be removed from this vector.

Table 26-119. Protocol Control Command Priorities

Protocol Command	Priority	Interrupted By	Cleared and Terminated By
FREEZE	(highest) 1	none	
READY	2		
CONFIG_COMPLETE	3		
ALL_SLOTS	4	FREEZE, READY, CONFIG_COMPLET, fatal protocol error	FREEZE, READY, CONFIG_COMPLETE, fatal protocol error
ALLOW_COLDSTART	5		
RUN	6		FREEZE, fatal protocol error
WAKEUP	7		FREEZE, fatal protocol error
DEFAULT_CONFIG	8		FREEZE, fatal protocol error
CONFIG	9		
HALT	(lowest) 10		FREEZE, READY, CONFIG_COMPLETE, fatal protocol error

26.7.5 Message Buffer Search on Simple Message Buffer Configuration

This sections describes the message buffer search behavior for a simplified message buffer configuration. The FIFO behavior is not considered in this section.

26.7.5.1 Simple Message Buffer Configuration

A simple message buffer configuration is a configuration that has at most one transmit message buffer and at most one receive message buffer assigned to a slot *S*. The simple configuration used in this section utilizes two message buffers, one single buffered transmit message buffer and one receive message buffer.

The transmit message buffer has the message buffer number *t* and has following configuration

Table 26-120. Transmit Buffer Configuration

Register	Field	Value	Description
MBCCSR*t*	MCM	-	used only for double buffers
	MBT	0	single transmit buffer
	MTD	1	transmit buffer
MBCCFR*t*	MTM	0	event transition mode
	CHA	1	assigned to channel A
	CHB	0	not assigned to channel B
	CCFE	1	cycle counter filter enabled
	CCFMSK	000011	cycle set = {4n} = {0,4,8,12,...}
	CCFVAL	000000	
MBFIDR*t*	FID	*S*	assigned to slot S

The availability of data in the transmit buffer is indicated by the commit bit MBCCSR*t*[CMT] and the lock bit MBCCSR*t*[LCKS].

The receive message buffer has the message buffer number *r* and has following configuration

Table 26-121. Receive Buffer Configuration

Register	Field	Value	Description
MBCCSR*r*	MCM	—	n/a
	MBT	—	n/a
	MTD	0	receive buffer
MBCCF*r*	MTM	—	n/a
	CHA	1	assigned to channel A
	CHB	0	not assigned to channel B
	CCFE	1	cycle counter filter enabled
	CCFMSK	000001	cycle set = {2n} = {0,2,4,6,...}
	CCFVAL	000000	
MBFIDR*r*	FID	*S*	subscribed slot

Furthermore the assumption is that both message buffers are enabled (MBCCSR*t*[EDS] = 1 and MBCCSR*r*[EDS] = 1)

NOTE

The cycle set {4n + 2} = {2,6,10,...} is assigned to the receive buffer only.

The cycle set {4n} = {0,4,8,12,...} is assigned to both buffers.

26.7.5.2 Behavior in Static Segment

In this case, both message buffers are assigned to a slot *S* in the *static* segment.

The configuration of a transmit buffer for a static slot *S* assigns this slot to the node as a transmit slot. The FlexRay protocol requires:

- When a slot occurs, if the slot is assigned to a node on a channel that node must transmit either a normal frame or a null frame on that channel. Specifically, a null frame will be sent if there is no data ready, or if there is no match on a transmit filter (cycle counter filtering, for example).

Regardless of the availability of data and the cycle counter filter, the node will transmit a frame in the static slot *S*. In any case, the result of the message buffer search will be the transmit message buffer *t*. The receive message buffer *r* will not be found, no reception is possible.

26.7.5.3 Behavior in Dynamic Segment

In this case, both message buffers are assigned to a slot *S* in the *dynamic* segment. The FlexRay protocol requires:

- When a slot occurs, if a slot is assigned to a node on a channel that node only transmits a frame on that channel if there is data ready and there is a match on relevant transmit filters (no null frames are sent).

The transmission of a frame in the dynamic segment is determined by the availability of data and the match of the cycle counter filter of the transmit message buffer.

26.7.5.3.1 Transmit Data Not Available

If transmit data are *not available*, i.e.,the transmit buffer is not committed MBCCSR*t*[CMT] = 0 and/or locked MBCCSR*t*[LCKS] = 1,

a) for the cycles in the set {4*n*}, which is assigned to both buffers, the receive buffer will be found and the node can receive data, and

b) for the cycles in the set {4*n* + 2}, which is assigned to the receive buffer only, the receive buffer will be found and the node can receive data.

The receive cycles are shown in Figure 26-153.

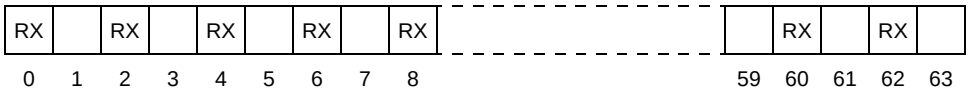


Figure 26-153. Transmit Data Not Available

26.7.5.3.2 Transmit Data Available

If transmit data are *available*, i.e.,the transmit buffer is committed MBCCSR*t*[CMT] = 1 and not locked MBCCSR*t*[LCKS] = 0,

a) for the cycles in the set {4n}, which is assigned to both buffers, the transmit buffer will be found and the node transmits data.

b) for the cycles in the set {4n + 2}, which is assigned to the receive buffer only, the receive buffer will be found and the node can receive data.

The receive and transmit cycles are shown in Figure 26-153

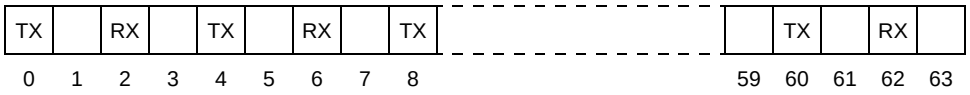


Figure 26-154. Transmit Data Not Available

Chapter 27
Media Local Bus (MLB)

27.1 Introduction

The MediaLB (MLB) is a multiplexed bus protocol defined by Standard Microsystems Semiconductor Company (SMSC) to transfer multimedia data between the Media-Oriented Systems Transport (MOST) ring and supporting system level ICs. It supports the complete MLB specification. This module offers a serial to parallel conversion of the 3-pin MediaLB signals into 32-bit parallel words and vice versa for transfer to system memory. This module provides a MediaLB port for all MediaLB relevant signals and an application port to interface to the PXN20. The application port incorporates all signals necessary to access MediaLB data bytes and protocol information as well as signals for controlling MediaLB Device Core functionality.

NOTE

The MLB block is not implemented on the PXN20.

27.1.1 Block Diagram

A block diagram of the MLB module can be found in Figure 27-1.

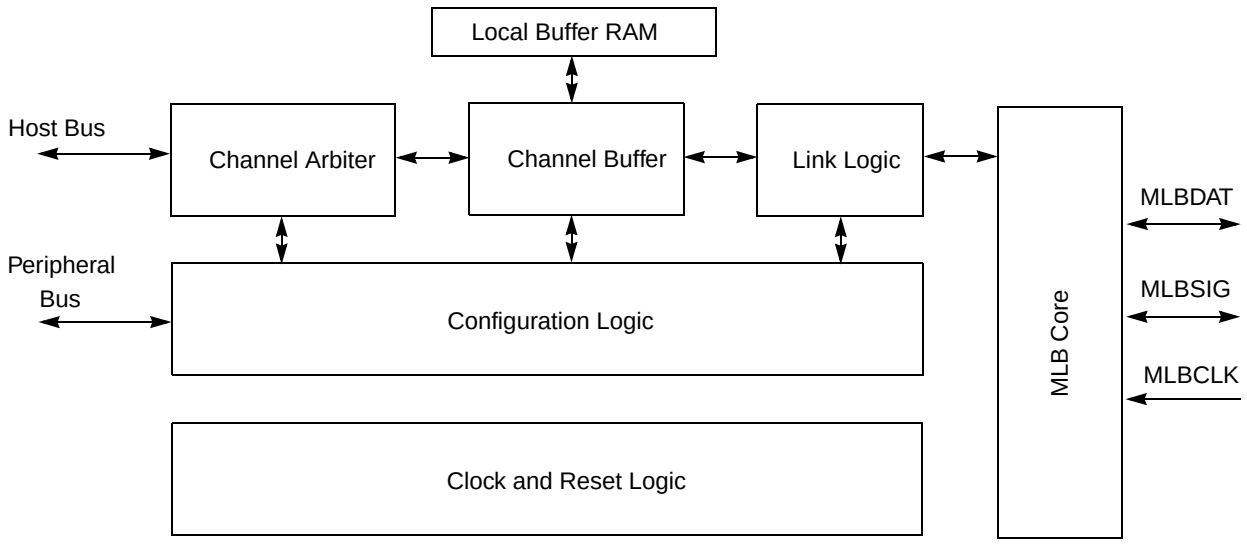


Figure 27-1. Block Diagram

27.1.2 Features

- 3-pin MediaLB interface supported
- Support for as many as 16 logical channels and as many as 31 physical channels running at a maximum speed of 1024Fs
- Programmable MediaLB clock frequency
  - 256 Fs = 12.3 megabits per second @ Fs = 48 kHz
  - 512 Fs = 24.6 megabits per second @ Fs = 48 kHz
- Supports all MOST data types:
  - Control messages
  - Packet messages (asynchronous)
  - Streaming data (synchronous)
  - Isochronous data (when supported by MLB master)
- Supports DMA style backend user interface
- Transmission of commands and data and reception of receive status when functioning as the transmitting device associated with a logical channel address
- Reception of commands and data and transmission as receive status responses when functioning as the receiving device associated with a logical channel address
- MLB lock detection
- System channel command handling
- System bus master supporting direct memory access between the MLB and system RAM
- Local channel buffer memory of 2K quadlets shared between all the logical channels

27.1.3 Overview

The MLB module implements the physical layer and link layer of the Media Local Bus specification, interfacing to the MLB controller. The MLB implements the 3-pin MLB mode and can run at speeds as fast as 1024Fs. It does not implement MLB controller functionality.

All MLB devices support a set of physical channels for sending data over the MLB. Each physical channel is 4 bytes in length (quadlet) and grouped into logical channels with one or more physical channels allocated to each logical channel. These logical channels can be any combination of channel type (synchronous, asynchronous, control, or isochronous) and direction (transmit or receive).

The MLB provides support for as many as 16 logical channels and as many as 31 physical channels with a maximum of 124 bytes of data per frame. Each logical channel is referenced using an unique channel address and represents a unidirectional data path between a MLB device transmitting the data and the MLB device(s) receiving the data.

Once per MOST network frame, the MLB controller generates a unique frame sync pattern. This pattern defines the frame and channel boundaries of the signal information and data lines.

The MLB controller initiates all communication over the MLB by sending out the logical channel address on the signal information line for each physical channel. This logical address indicates to the appropriate

MLB device that it can transmit data for that logical channel during the next physical channel slot. One quadlet later, the transmitting MLB device send out a MLB command byte on the signal information line and the corresponding data on the data line. All other MLB devices (including the controller) have already compared the logical channel address with their internal table of addresses to determine if they are the intended recipient of the data on this logical channel.

The receiving MLB device responds with a receive status response on the signal information line one byte after the transmitting device sends the MLB command byte. Note that synchronous data transmissions (which is the only data format to support multiple receivers) are not acknowledged, but asynchronous, control and isochronous data transmissions are acknowledged.

For more information about the Media Local Bus protocol, please refer to the *Media Local Bus Specification*.

27.1.4 Modes of Operation

The byte order in which data is transferred between the MLB bus and the MLB device is always Big Endian; however, a multiplexor at the local channel buffer allows software to select the data byte order of data within system memory. Big Endian or Little Endian is selected via a software programmable configuration bit DCCR[MLE].

Table 27-1 illustrates the Big Endian data format. Table 27-2 illustrates the Little Endian data format.

Table 27-1. Big Endian Byte Order

Address	data[31:24]	data[23:16]	data[15:8]	data[7:0]
0x0000_0000	Byte 00	Byte 01	Byte 02	Byte 03
0x0000_0004	Byte 04	Byte 05	Byte 06	Byte 07
0x0000_0008	Byte 08	Byte 09	Byte 10	Byte 11
0x0000_000C	Byte 12	Byte 13	Byte 14	Byte 15

Table 27-2. Little Endian Byte Order

Address	data[31:24]	data[23:16]	data[15:8]	data[7:0]
0x0000_0000	Byte 03	Byte 02	Byte 01	Byte 00
0x0000_0004	Byte 07	Byte 06	Byte 05	Byte 04
0x0000_0008	Byte 11	Byte 10	Byte 09	Byte 08
0x0000_000C	Byte 15	Byte 14	Byte 13	Byte 12

27.2 External Signal Description

The MLB peripheral contains three external pins to interface to the MLB controller. They are shown in Table 27-3.

Table 27-3. Signal Properties

Signal	Port	SIU_PCR Register	Function	I/O	Reset	Pull
MLBCLK	PK0	SIU_PCR144	MLB Clock	I	0	Down
MLBSIG	PK1	SIU_PCR145	MLB Signal (control/status)	I/O	0	Down
MLBDAT	PK2	SIU_PCR146	MLB Data	I/O	0	Down

Detailed signal descriptions for the MLB peripheral can be found in Table 27-4.

Table 27-4. MLB—Detailed Signal Descriptions

Signal	I/O	Description	
MLBCLK	I	MLB Clock.	
		State Meaning	Asserted/Negated—Supports a 256Fs, 512Fs or 1024Fs clock input from the MLB controller.
		Timing	Assertion/Negation—Supports maximum frequency of 49.2 MHz with a 48 kHz sample rate.
MLBDAT	I/O	MLB Data	
		State Meaning	Asserted/Negated—MLB data for serial receive/transmit channel data.
		Timing	Assertion/Negation—Input registered on the falling edge of MLBCLK. Output driven from the rising edge of MLBCLK.
MLBSIG	I/O	MLB Signal (control/status)	
		State Meaning	Asserted/Negated—MLB signal information for serial transmit channel commands, serial receive channel responses, and logical channel address information.
		Timing	Assertion/Negation—Input registered on the falling edge of MLBCLK. Output driven from the rising edge of MLBCLK.

27.3 Memory Map and Register Description

27.3.1 Memory Map

Table 27-5 shows the MLB configuration registers. Table 27-6 shows the channel configuration registers for each channel within the MLB. Table 27-7 shows the overall memory map for the MLB.

Table 27-5. Configuration Registers

Offset from MLB_BASE (0xC3F8_4000)	Name	Access
0x0000_0000	DCCR—Device Control Configuration Register	R/W
0x0000_0004	SSCR—System Status Configuration Register	R/W
0x0000_0008	SDCR—System Data Configuration Register	R
0x0000_000C	SMCR—System Mask Configuration Register	R/W

Table 27-5. Configuration Registers (continued)

Offset from MLB_BASE (0xC3F8_4000)	Name	Access
0x0000_001C	VCCR—Version Control Configuration Register	R
0x0000_0020	SBCR—Synchronous Base Address Configuration Register	R/W
0x0000_0024	ABCR—Asynchronous Base Address Configuration Register	R/W
0x0000_0028	CBCR—Control Base Address Configuration Register	R/W
0x0000_002C	IBCR—Isochronous Base Address Configuration Register	R/W
0x0000_0030	CICR—Channel Interrupt Configuration Register	R

Table 27-6. Channel Configuration Registers

Offset from MLB_BASE (0xC3F8_4000)	Channel *n* Register (*n* = 0..15)	Access
0x0010 + *n* × 0x10	CECR*n*—Channel *n* Entry Configuration Register	R/W
0x0014 + *n* × 0x10	CSCR*n*—Channel *n* Status Configuration Register	R/W
0x0018 + *n* × 0x10	CCBCR*n*—Channel *n* Current Buffer Configuration Register	R
0x001A + *n* × 10	CNBCR*n*—Channel *n* Next Buffer Configuration Register	R/W
0x0280 + *n* × 0x04	LCBCR*n*—Local Channel *n* Buffer Configuration Register	R/W

Table 27-7. MLB Memory Map

Offset from MLB_BASE (0xC3F8_4000)	Register	Access	Reset Value	Section/Page
0x0000	DCCR—Device Control Configuration Register	R/W[1]	0x0000_0000	27.3.2.1/27-8
0x0004	SSCR—System Status Configuration Register	R/W	0x0000_0000	27.3.2.2/27-10
0x0008	SDCR—System Data Configuration Register	R	0x0000_0000	27.3.2.3/27-11
0x000C	SMCR—System Mask Configuration Register	R/W	0x0000_0060	27.3.2.4/27-12
0x001C	VCCR—Version Control Configuration Register	R	0x0300_0202	27.3.2.5/27-13
0x0020	SBCR—Synchronous Base Address Configuration Register	R/W	0x0000_0000	27.3.2.6/27-14
0x0024	ABCR—Asynchronous Base Address Configuration Register	R/W	0x0000_0000	27.3.2.7/27-14
0x0028	CBCR—Control Base Address Configuration Register	R/W	0x0000_0000	27.3.2.8/27-15
0x002C	IBCR—Isochronous Base Address Configuration Register	R/W	0x0000_0000	27.3.2.9/27-16
0x0030	CICR—Channel Interrupt Configuration Register	R	0x0000_0000	27.3.2.10/27-16
0x0040	CECR0—Channel 0 Entry Configuration Register	R/W	0x0000_0000	27.3.2.11/27-17
0x0044	CSCR0—Channel 0 Status Configuration Register	R/W	0x8000_0000	27.3.2.12/27-19

Table 27-7. MLB Memory Map (continued)

Offset from MLB_BASE (0xC3F8_4000)	Register	Access	Reset Value	Section/Page
0x0048	CCBCR0—Channel 0 Current Buffer Configuration Register	R/W	0x0000_0000	27.3.2.13/27-22
0x004C	CNBCR0—Channel 0 Next Buffer Configuration Register	R/W	0x0000_0000	27.3.2.14/27-23
0x0050	CECR1—Channel 1 Entry Configuration Register	R/W	0x0000_0000	27.3.2.11/27-17
0x0054	CSCR1—Channel 1 Status Configuration Register	R/W	0x8000_0000	27.3.2.12/27-19
0x0058	CCBCR1—Channel 1 Current Buffer Configuration Register	R/W	0x0000_0000	27.3.2.13/27-22
0x005C	CNBCR1—Channel 1 Next Buffer Configuration Register	R/W	0x0000_0000	27.3.2.14/27-23
0x0060	CECR2—Channel 2 Entry Configuration Register	R/W	0x0000_0000	27.3.2.11/27-17
0x0064	CSCR2—Channel 2 Status Configuration Register	R/W	0x8000_0000	27.3.2.12/27-19
0x0068	CCBCR2—Channel 2 Current Buffer Configuration Register	R/W	0x0000_0000	27.3.2.13/27-22
0x006C	CNBCR2—Channel 2 Next Buffer Configuration Register	R/W	0x0000_0000	27.3.2.14/27-23
0x0070	CECR3—Channel 3 Entry Configuration Register	R/W	0x0000_0000	27.3.2.11/27-17
0x0074	CSCR3—Channel 3 Status Configuration Register	R/W	0x8000_0000	27.3.2.12/27-19
0x0078	CCBCR3—Channel 3 Current Buffer Configuration Register	R/W	0x0000_0000	27.3.2.13/27-22
0x007C	CNBCR3—Channel 3 Next Buffer Configuration Register	R/W	0x0000_0000	27.3.2.14/27-23
0x0080	CECR4—Channel 4 Entry Configuration Register	R/W	0x0000_0000	27.3.2.11/27-17
0x0084	CSCR4—Channel 4 Status Configuration Register	R/W	0x8000_0000	27.3.2.12/27-19
0x0088	CCBCR4—Channel 4 Current Buffer Configuration Register	R/W	0x0000_0000	27.3.2.13/27-22
0x008C	CNBCR4—Channel 4 Next Buffer Configuration Register	R/W	0x0000_0000	27.3.2.14/27-23
0x0090	CECR5—Channel 5 Entry Configuration Register	R/W	0x0000_0000	27.3.2.11/27-17
0x0094	CSCR5—Channel 5 Status Configuration Register	R/W	0x8000_0000	27.3.2.12/27-19
0x0098	CCBCR5—Channel 5 Current Buffer Configuration Register	R/W	0x0000_0000	27.3.2.13/27-22
0x009C	CNBCR5—Channel 5 Next Buffer Configuration Register	R/W	0x0000_0000	27.3.2.14/27-23
0x00A0	CECR6—Channel 6 Entry Configuration Register	R/W	0x0000_0000	27.3.2.11/27-17
0x00A4	CSCR6—Channel 6 Status Configuration Register	R/W	0x8000_0000	27.3.2.12/27-19
0x00A8	CCBCR6—Channel 6 Current Buffer Configuration Register	R/W	0x0000_0000	27.3.2.13/27-22
0x00AC	CNBCR6—Channel 6 Next Buffer Configuration Register	R/W	0x0000_0000	27.3.2.14/27-23
0x00B0	CECR7—Channel 7 Entry Configuration Register	R/W	0x0000_0000	27.3.2.11/27-17
0x00B4	CSCR7—Channel 7 Status Configuration Register	R/W	0x8000_0000	27.3.2.12/27-19
0x00B8	CCBCR7—Channel 7 Current Buffer Configuration Register	R/W	0x0000_0000	27.3.2.13/27-22
0x00BC	CNBCR7—Channel 7 Next Buffer Configuration Register	R/W	0x0000_0000	27.3.2.14/27-23
0x00C0	CECR8—Channel 8 Entry Configuration Register	R/W	0x0000_0000	27.3.2.11/27-17

Table 27-7. MLB Memory Map (continued)

Offset from MLB_BASE (0xC3F8_4000)	Register	Access	Reset Value	Section/Page
0x00C4	CSCR8—Channel 8 Status Configuration Register	R/W	0x8000_0000	27.3.2.12/27-19
0x00C8	CCBCR8—Channel 8 Current Buffer Configuration Register	R/W	0x0000_0000	27.3.2.13/27-22
0x00CC	CNBCR8—Channel 8 Next Buffer Configuration Register	R/W	0x0000_0000	27.3.2.14/27-23
0x00D0	CECR9—Channel 9 Entry Configuration Register	R/W	0x0000_0000	27.3.2.11/27-17
0x00D4	CSCR9—Channel 9 Status Configuration Register	R/W	0x8000_0000	27.3.2.12/27-19
0x00D8	CCBCR9—Channel 9 Current Buffer Configuration Register	R/W	0x0000_0000	27.3.2.13/27-22
0x00DC	CNBCR9—Channel 9 Next Buffer Configuration Register	R/W	0x0000_0000	27.3.2.14/27-23
0x00E0	CECR10—Channel 10 Entry Configuration Register	R/W	0x0000_0000	27.3.2.11/27-17
0x00E4	CSCR10—Channel 10 Status Configuration Register	R/W	0x8000_0000	27.3.2.12/27-19
0x00E8	CCBCR10—Channel 10 Current Buffer Configuration Register	R/W	0x0000_0000	27.3.2.13/27-22
0x00EC	CNBCR10—Channel 10 Next Buffer Configuration Register	R/W	0x0000_0000	27.3.2.14/27-23
0x00F0	CECR11—Channel 11 Entry Configuration Register	R/W	0x0000_0000	27.3.2.11/27-17
0x00F4	CSCR11—Channel 11 Status Configuration Register	R/W	0x8000_0000	27.3.2.12/27-19
0x00F8	CCBCR11—Channel 11 Current Buffer Configuration Register	R/W	0x0000_0000	27.3.2.13/27-22
0x00FC	CNBCR11—Channel 11 Next Buffer Configuration Register	R/W	0x0000_0000	27.3.2.14/27-23
0x0100	CECR12—Channel 12 Entry Configuration Register	R/W	0x0000_0000	27.3.2.11/27-17
0x0104	CSCR12—Channel 12 Status Configuration Register	R/W	0x8000_0000	27.3.2.12/27-19
0x0108	CCBCR12—Channel 12 Current Buffer Configuration Register	R/W	0x0000_0000	27.3.2.13/27-22
0x010C	CNBCR12—Channel 12 Next Buffer Configuration Register	R/W	0x0000_0000	27.3.2.14/27-23
0x0110	CECR13—Channel 13 Entry Configuration Register	R/W	0x0000_0000	27.3.2.11/27-17
0x0114	CSCR13—Channel 13 Status Configuration Register	R/W	0x8000_0000	27.3.2.12/27-19
0x0118	CCBCR13—Channel 13 Current Buffer Configuration Register	R/W	0x0000_0000	27.3.2.13/27-22
0x011C	CNBCR13—Channel 13 Next Buffer Configuration Register	R/W	0x0000_0000	27.3.2.14/27-23
0x0120	CECR14—Channel 14 Entry Configuration Register	R/W	0x0000_0000	27.3.2.11/27-17
0x0124	CSCR14—Channel 14 Status Configuration Register	R/W	0x8000_0000	27.3.2.12/27-19
0x0128	CCBCR14—Channel 14 Current Buffer Configuration Register	R/W	0x0000_0000	27.3.2.13/27-22
0x012C	CNBCR14—Channel 14 Next Buffer Configuration Register	R/W	0x0000_0000	27.3.2.14/27-23
0x0130	CECR15—Channel 15 Entry Configuration Register	R/W	0x0000_0000	27.3.2.11/27-17
0x0134	CSCR15—Channel 15 Status Configuration Register	R/W	0x8000_0000	27.3.2.12/27-19
0x0138	CCBCR15—Channel 15 Current Buffer Configuration Register	R/W	0x0000_0000	27.3.2.13/27-22
0x013C	CNBCR15—Channel 15 Next Buffer Configuration Register	R/W	0x0000_0000	27.3.2.14/27-23

Table 27-7. MLB Memory Map (continued)

Offset from MLB_BASE (0xC3F8_4000)	Register	Access	Reset Value	Section/Page
0x0140–0x027F	Reserved			
0x0280	LCBCR0—Local Channel 0 Buffer Configuration Register	R/W	0x0803_E000	27.3.2.15/27-24
0x0284	LCBCR1—Local Channel 1 Buffer Configuration Register	R/W	0x0803_E020	27.3.2.15/27-24
0x0288	LCBCR2—Local Channel 2 Buffer Configuration Register	R/W	0x0803_E040	27.3.2.15/27-24
0x028C	LCBCR3—Local Channel 3 Buffer Configuration Register	R/W	0x0803_E060	27.3.2.15/27-24
0x0290	LCBCR4—Local Channel 4 Buffer Configuration Register	R/W	0x0803_E080	27.3.2.15/27-24
0x0294	LCBCR5—Local Channel 5 Buffer Configuration Register	R/W	0x0803_E0A0	27.3.2.15/27-24
0x0298	LCBCR6—Local Channel 6 Buffer Configuration Register	R/W	0x0803_E0C0	27.3.2.15/27-24
0x029C	LCBCR7—Local Channel 7 Buffer Configuration Register	R/W	0x0803_E0E0	27.3.2.15/27-24
0x02A0	LCBCR8—Local Channel 8 Buffer Configuration Register	R/W	0x0803_E100	27.3.2.15/27-24
0x02A4	LCBCR9—Local Channel 9 Buffer Configuration Register	R/W	0x0803_E120	27.3.2.15/27-24
0x02A8	LCBCR10—Local Channel 10 Buffer Configuration Register	R/W	0x0803_E140	27.3.2.15/27-24
0x02AC	LCBCR11—Local Channel 11 Buffer Configuration Register	R/W	0x0803_E160	27.3.2.15/27-24
0x02B0	LCBCR12—Local Channel 12 Buffer Configuration Register	R/W	0x0803_E180	27.3.2.15/27-24
0x02B4	LCBCR13—Local Channel 13 Buffer Configuration Register	R/W	0x0803_E1A0	27.3.2.15/27-24
0x02B8	LCBCR14—Local Channel 14 Buffer Configuration Register	R/W	0x0803_E1C0	27.3.2.15/27-24
0x02BC	LCBCR15—Local Channel 15 Buffer Configuration Register	R/W	0x0803_E1E0	27.3.2.15/27-24
0x02C0–0x3FFF	Reserved			

[1] Note that R/W registers may contain some read-only or write-only bits.

27.3.2 Register Descriptions

27.3.2.1 Device Control Configuration Register (DCCR)

The Device Control Configuration Register (DCCR) is used to control basic features of the MLB device, such as clock rate, lock status, enable, and reset behavior.

Offset: MLB_BASE + 0x0000 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	MDE	LBM	MCS[1:0]		0	MLK	MLE	MHRE	MRS	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	MDA[8:1]							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 27-2. Device Control Configuration Register (DCCR)

Table 27-8. DCCR Field Descriptions

Field	Description
MDE	MLB Device Enable. When set, enables the MLB Interface based on the other bits in the register. 0 MLB device disabled. 1 MLB device enabled.
LBM	Loop-Back Mode Enable. When set, enables the loop-back testing of the MLB bus between logical channel 0 (RX) and logical channel 1 (TX). 0 Loop-back mode disabled. 1 Loop-Back Mode Enabled.
MCS [1:0]	MLB Clock Select. These field must be programmed by system software to reflect the MLBCLK speed. 00 256Fs: supports 8 quadlets per frame. 01 512Fs: supports 16 quadlets per frame. 10 1024Fs: supports 32 quadlets per frame. 11 Reserved.
MLK	MLB Lock. When set, indicates that the MLB Port is synchronized to the incoming MLB frame. If MLK is clear (unlocked), MLK is set after FRAMESYNC is detected at the same position for three consecutive frames. If MLK is set (locked), MLK is cleared after not receiving FRAMESYNC at the expected time for two consecutive frames. While MLK is set, FRAMESYNC patterns occurring at locations other than the expected one are ignored. 0 MLB device is not synchronized to the incoming MLB frame. 1 MLB device is synchronized to the incoming MLB frame.
MLE	MLB Little Endian. This field determines how MLB quadlet based data is stored in system memory. For normal operation, this bit should be set. 0 Little Endian mode. 1 Big Endian mode.
MHRE	MLB Hardware Reset Enable. When set, enables hardware to automatically reset the MLB physical and link layer logic upon the reception of either a global (SDCR[MDS] = 8'h0000) or device specific (SDCR[MDS] = DA) MlbReset (FEh) System Command. 0 MLB device does not reset on reception of system reset command. 1 MLB device is reset on reception of system reset command.

Table 27-8. DCCR Field Descriptions (continued)

Field	Description
MRS	MLB Software Reset. When set, resets the MLB physical and link layer logic. Hardware clears this bit automatically. 0 MLB device is not reset by software. 1 MLB device is reset by software.
MDA [8:1]	MLB Device Address. Determines the unique *DeviceAddress* (DA) for the MLB Device. *DeviceAddresses* are used by the system channel *MlbScan* command. DA[15:0] = { 7'h00, MDA[8:1], 1'b0 }

27.3.2.2 System Status Configuration Register (SSCR)

The System Status Configuration Register (SSCR) allows system software to monitor and control the status of the MLB network. SSCR is updated once per frame by hardware during the MLB System Channel. Except for the bits associated with MLB lock and unlock (SSCR[SDMU] and SSCR[SDML]), the bits of the SSCR register are not valid until the MLB is locked to the MLB interface. System software must service status events before the start of the next MLB frame to prevent the current frame status from being lost.

Offset: MLB_BASE + 0x0004 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	SSRE	SDMU	SDML	SDSC	SDCS	SDNU	SDNL	SDR
W										w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 27-3. System Status Configuration Register (SSCR)

Table 27-9. SSCR Field Descriptions

Field	Description
SSRE	System Service Request Enable. System software can set this bit to indicate that this MLB Device is present and needs service. An RxStatus *DeviceServiceRequest* (82h) is sent in response to a *MlbScan* System Command from the MLB Controller. Hardware clears this bit after the RxStatus is sent. 0 MLB device responds to System Scan Command with Device Present (80h). 1 MLB device responds to System Scan Command with Device Service Request (82h).
SDMU	System Detects MLB Unlock. This bit is set to indicate that the MLB Device has unlocked from the MLB frame. Detecting a MLB unlock generates a maskable system interrupt to system software. Once set, this bit is sticky until cleared by software by writing a logic one to this bit. 0 MLB device has not unlocked from MLB frame. 1 MLB device has unlocked from MLB frame.

Table 27-9. SSCR Field Descriptions (continued)

Field	Description
SDML	System Detects MLB Lock. This bit is set to indicate that the MLB Device has locked to the MLB frame. Detecting a MLB lock generates a maskable system interrupt to system software. Once set, this bit is sticky until cleared by software. 0 MLB device has not locked to MLB frame. 1 MLB device has locked to MLB frame.
SDSC	System Detects SubCommand. This bit is set to indicate that the MLB Device has received the *MlbSubCmd* (E6h) System Command. The user-defined software command is stored in the SDCR register. The decoding of this command is left up to software. Detecting *MlbSubCmd* generates a maskable system interrupt to system software. Once set, this bit is sticky until cleared by software. 0 MLB device has not detected a Sub-command System Command. 1 MLB device has detected a Sub-command System Command.
SDCS	System Detects Channel Scan. This bit is set to indicate that the MLB Device has received the *MlbScan* (E4h) System Command. The target *DeviceAddress* is stored in the SDCR register. Detecting *MlbScan* generates a maskable system interrupt to system software. Once set, this bit is sticky until cleared by software. 0 MLB device has not detected a System Scan Command. 1 MLB device has detected a System Scan Command.
SDNU	System Detects Network Unlock. This bit is set to indicate that the MLB Device has received the *MOST_Unlock* (E2h) System Command. Detecting *MOST_Unlock* generates a maskable system interrupt to system software. Once set, this bit is sticky until cleared by software. 0 MLB device has not detected an Unlock Command. 1 MLB device has detected an Unlock Command.
SDNL	System Detects Network Lock. This bit is set to indicate that the MLB Device has received the *MOST_Lock* (E0h) System Command. Detecting *MOST_Lock* generates a maskable system interrupt to system software. Once set, this bit is sticky until cleared by software. 0 MLB device has not detected a Lock Command. 1 MLB device has detected a Lock Command.
SDR	System Detects Reset. This bit is set to indicate that the MLB Device has received the *MlbReset* (FEh) System Command. The target *DeviceAddress* is stored in the SDCR register. Detecting *MLBReset* generates a maskable system interrupt to system software. Once set, this bit is sticky until cleared by software. 0 MLB device has not detected a Reset Command. 1 MLB device has detected a Reset Command.

27.3.2.3 System Data Configuration Register (SDCR)

The System Data Configuration Register (SDCR) allows system software to receive control information from the MLB Controller. SDCR is updated once per frame by hardware during the MLB System Channel. System software must read SDCR before the start of the next MLB frame to prevent the current frame data from being lost.

Offset: MLB_BASE + 0x0008 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	MSD[31:16]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	MSD[15:0]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 27-4. System Data Configuration Register (SDCR)

Table 27-10. SDCR Field Descriptions

Field	Description
MSD	MLB System Data. This register is loaded with the data from MLBDAT during the System Channel quadlet.

27.3.2.4 System Mask Configuration Register (SMCR)

The System Mask Configuration register (SMCR) allows system software to mask system status interrupts.

Offset: MLB_BASE + 0x000C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	SMMU	SMML	SMSC	SMCS	SMNU	SMNL	SMR
W																
Reset	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0

Figure 27-5. System Mask Configuration Register (SMCR)

Table 27-11. SMCR Field Descriptions

Field	Description
SMMU	System Masks MLB Unlock. When set, this bit masks system interrupts generated when a MLB unlock is detected. At reset, MLB unlock events are masked (SMMU = 1). 0 MLB unlock system interrupt is enabled. 1 MLB unlock system interrupt is disabled.
SMML	System Masks MLB Lock. When set, this bit masks system interrupts generated when MLB lock is detected. At reset, MLB lock events are masked (SMML = 1). 0 MLB lock system interrupt is enabled. 1 MLB lock system interrupt is disabled.
SMSC	System Masks SubCommand. When set, this bit masks system interrupts for the *MlbSubCmd* (E6h) System Command. 0 MLB SubCommand system interrupt is enabled. 1 MLB SubCommand system interrupt is disabled.
SMCS	System Masks Channel Scan. When set, this bit masks system interrupts for the *MlbScan* (E4h) System Command. 0 MLB Channel Scan system interrupt is enabled. 1 MLB Channel Scan system interrupt is disabled.
SMNU	System Masks Network Unlock. When set, this bit masks system interrupts for the *MOST_Unlock* (E2h) System Command. 0 MLB Network Unlock system interrupt is enabled. 1 MLB Network Unlock system interrupt is disabled.
SMNL	System Masks Network Lock. When set, this bit masks system interrupts for the *MOST_Lock* (E0h) System Command. 0 MLB Network Lock system interrupt is enabled. 1 MLB Network Lock system interrupt is disabled.
SMR	System Masks Reset. When set, this bit masks system interrupts for the *MlbReset* (FEh) System Command. 0 MLB Reset system interrupt is enabled. 1 MLB Reset system interrupt is disabled.

27.3.2.5 Version Control Configuration Register (VCCR)

The Version Control Configuration Register (VCCR) allows system software to verify the version of the MLB and the user implementation.

Offset: MLB_BASE + 0x001C Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	UMA[7:0]								UMI[7:0]							
W																
Reset	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	MMA[7:0]								MMI[7:0]							
W																
Reset	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0

Figure 27-6. Version Control Configuration Register (VCCR)

Table 27-12. VCCR Field Descriptions

Field	Description
UMA [7:0]	User Major Revision. For first release of the PXN20, the value is 0x03.
UMI [7:0]	User Minor Revision. For first release of the PXN20, the value is 0x00.
MMA [7:0]	MLB Device Major Revision. For first release of the PXN20, the value is 0x02.
MMI [7:0]	MLB Device Minor Revision.For first release of the PXN20, the value is 0x02.

27.3.2.6 Synchronous Base Address Configuration Register (SBCR)

The Synchronous Base Address Configuration Register (SBCR) allows system software to define the base address for synchronous RX/TX system memory buffers.

Offset: MLB_BASE + 0x0020 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	SRBA[31:16]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	STBA[31:16]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 27-7. Synchronous Base Address Configuration Register (SBCR)

Table 27-13. SSBCR Field Descriptions

Field	Description
SRBA [31:16]	Synchronous Receive Base Address. This base address is shared by all synchronous RX channels and defines the upper 16 bits of the 32-bit system memory address for these channels.
STBA [31:16]	Synchronous Transmit Base Address. This base address is shared by all synchronous TX channels and defines the upper 16 bits of the 32-bit system memory address for these channels.

27.3.2.7 Asynchronous Base Address Configuration Register (ABCR)

The Asynchronous Base Address Configuration Register (ABCR) allows system software to define the base address for asynchronous RX/TX system memory buffers.

Offset: MLB_BASE + 0x0024 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	ARBA[31:16]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	ATBA[31:16]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 27-8. Asynchronous Base Address Configuration Register (ABCR)

Table 27-14. ABCR Field Descriptions

Field	Description
ARBA [31:16]	Asynchronous Receive Base Address. This base address is shared by all asynchronous RX channels and defines the upper 16 bits of the 32-bit system memory address for these channels.
ATBA [31:16]	Asynchronous Transmit Base Address. This base address is shared by all asynchronous TX channels and defines the upper 16 bits of the 32-bit system memory address for these channels.

27.3.2.8 Control Base Address Configuration Register (CBCR)

The Control Base Address Configuration Register (CBCR) allows system software to define the base address for control RX/TX system memory buffers.

Offset: MLB_BASE + 0x0028 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	CRBA[31:16]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	CTBA[31:16]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 27-9. Control Base Address Configuration Register (CBCR)

Table 27-15. CBCR Field Descriptions

Field	Description
CRBA [31:16]	Control Receive Base Address. This base address is shared by all control RX channels and defines the upper 16 bits of the 32-bit system memory address for these channels.
CTBA [31:16]	Control Transmit Base Address. This base address is shared by all control TX channels and defines the upper 16 bits of the 32-bit system memory address for these channels.

27.3.2.9 Isochronous Base Address Configuration Register (IBCR)

The Isochronous Base Address Configuration Register (IBCR) allows system software to define the base address for isochronous RX/TX system memory buffers.

Offset: MLB_BASE + 0x002C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	IRBA[31:16]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	ITBA[31:16]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 27-10. Isochronous Base Address Configuration Register (IBCR)

Table 27-16. IBCR Field Descriptions

Field	Description
IRBA [31:16]	Isochronous Receive Base Address. This base address is shared by all Isochronous RX channels and defines the upper 16 bits of the 32-bit system memory address for these channels.
ITBA [31:16]	Isochronous Transmit Base Address. This base address is shared by all Isochronous TX channels and defines the upper 16 bits of the 32-bit system memory address for these channels.

27.3.2.10 Channel Interrupt Configuration Register (CICR)

The Channel Interrupt Configuration Register (CICR) reflects the channel interrupt status of the individual MLB logical channels. These bits are set by hardware when a channel interrupt is generated. The channel interrupt bits are sticky and can only be reset by software.

Offset: MLB_BASE + 0x0030 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CSU[15:0]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 27-11. Channel Interrupt Configuration Register (CICR)

Table 27-17. CICR Field Descriptions

Field	Description
CSU [15:0]	Channel Status Update for Logical Channels 15 through 0. When set, these bits indicate that hardware has generated an interrupt for the appropriate channel. These bits are sticky and can only be cleared by a software write. Writing to the CICR register has no effect. To clear a particular bit in the CICR, software must clear all of the unmasked status bits in the corresponding CSCR*n* register. 0 Channel *n* has not generated an interrupt. 1 Channel *n* has generated an interrupt.

27.3.2.11 Channel *n* Entry Configuration Register

The Channel *n* Entry Configuration Register (CECR*n*) defines basic attributes about a given logical channel, such as the channel enable, channel type, channel direction, and channel address. The definitions of some of the bit fields in the CECR*n* register vary depending on the selected channel type.

Offset: 0x0040 (CECR0) 0x0080 (CECR4) 0x00C0 (CECR8) 0x0100 (CECR12) Access: User read/write
0x0050 (CECR1) 0x0090 (CECR5) 0x00D0 (CECR9) 0x0110 (CECR13)
0x0060 (CECR2) 0x00A0 (CECR6) 0x00E0 (CECR10) 0x0120 (CECR14)
0x0070 (CECR3) 0x00B0 (CECR7) 0x00F0 (CECR11) 0x0130 (CECR15)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CE	TR	CT[1:0]		FSE/ FCE	MDS[1:0]		0	0	MLFS	0	MBE	MBS	MBD	MDB	MPE
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	FSCD IPL[7]	IPL[6:5]			FSPC[4:0] / IPL[4:0]				CA[8:1]							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 27-12. Channel *n* Entry Configuration Register (CECR*n*)

Table 27-18. CECR*n* Field Descriptions

Field	Description
CE	Channel *n* Enable. 0 Channel *n* disabled. 1 Channel *n* enabled.
TR	Channel *n* Transmit Select. 0 Receive. 1 Transmit.
CT[1:0]	Channel *n* Type Select. 00 Synchronous. 01 Isochronous. 10 Asynchronous. 11 Control.
FSE	Frame Synchronization Enable. When set, enables *Streaming Channel Frame Synchronization* for this logical synchronous channel. This field is valid for synchronous channels only. 0 Disable frame synchronization for synchronous channels. 1 Enable frame synchronization for synchronous channels.
FCE	Flow Control Enable. When set, allows an isochronous RX channel to generate the *ReceiverBusy* (10h) response. This field is valid for isochronous channels only. 0 Disable flow control for isochronous channels. 1 Enable flow control for isochronous channels.
MDS [1:0]	Channel *n* Mode Select. 00 DMA Ping-pong buffering enabled. 01 DMA Circular buffering enabled. 10 Reserved. 11 Reserved.
MLFS	Mask Lost Frame Synchronization. When set, masks *Lost Frame Synchronization* channel interrupts for this logical channel. 0 Enable *Lost Frame Synchronization* channel interrupts for this logical channel. 1 Disable *Lost Frame Synchronization* channel interrupts for this logical channel.
MBE	Mask Buffer Error. When set, masks *Buffer Error* channel interrupts for this logical channel. 0 Enable *Buffer Error* channel interrupts for this logical channel. 1 Disable *Buffer Error* channel interrupts for this logical channel.
MBS	Mask Buffer Start. When set, masks *Buffer Start* channel interrupts for this logical channel. 0 Enable *Buffer Start* channel interrupts for this logical channel. 1 Disable *Buffer Start* channel interrupts for this logical channel.
MBD	Mask Buffer Done. When set, masks *Buffer Done* channel interrupts for this logical channel. 0 Enable *Buffer Done* channel interrupts for this logical channel. 1 Disable *Buffer Done* channel interrupts for this logical channel.
MDB	Mask Detect Break. When set, masks detect break channel interrupts for this logical channel. This bit is valid for asynchronous and control channels only. 0 Enable detect break channel interrupts for this logical channel. 1 Disable detect break channel interrupts for this logical channel.
MPE	Mask Protocol Error. When set, masks Protocol error channel interrupts for this logical channel. This bit is valid for all RX channel types and valid for only asynchronous and control TX channels. 0 Enable protocol error channel interrupts for this logical channel. 1 Disable protocol error channel interrupts for this logical channel.

Table 27-18. CECR*n* Field Descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>FSCD</td><td>Frame Synchronization Channel Disable. When set, disables this logical channel (set CECHRn[CE] = 0) when Lost Frame Synchronization occurs. This field is valid for synchronous channels only.
0 Do not disable this logical channel when frame synchronization is lost.
1 Disable this logical channel when frame synchronization is lost.</td></tr>
<tr><td>IPL
[7:0]</td><td>Isochronous Packet Length. For Isochronous TX channels, defines the number of packet bytes. The smallest isochronous packet size per frame is 5 bytes (IPL[7:0] ≥ 5). A packet length of 256 bytes can be represented as IPL[7:0] = 0x00. For Isochronous RX channels, software must program IPL[7:2] to indicate the expected number of bytes per packet, where IPL[1:0] always equals 0b00. A packet length of 253 bytes to 256 bytes can be represented as IPL[7:0] = 0x00. This field is valid for isochronous channels only.
<table>
<tr><th>IPL[7:0][1]</th><th>Bytes Per Packet (Expected)</th></tr>
<tr><td>0x00</td><td>253–256</td></tr>
<tr><td>0x08</td><td>5–8</td></tr>
<tr><td>0x0C</td><td>9–12</td></tr>
<tr><td>0x10</td><td>13–16</td></tr>
<tr><td>0x14 + 4n
n = 0 to 38</td><td>...</td></tr>
<tr><td>0xF8</td><td>245–248</td></tr>
<tr><td>0xFC</td><td>249–252</td></tr>
</table>
1 All other values are reserved.</td></tr>
<tr><td>FSPC
[4:0]</td><td>Frame Synchronization Physical Channel Count. Defines the number of physical channels expected to match this logical channel's ChannelAddress each MediaLB frame. This field is valid for synchronous channels only.</td></tr>
<tr><td>CA[8:1]</td><td>Channel Address. These bits determine the ChannelAddress associated with this logical channel. This value is matched against the ChannelAddress received each physical channel from the MediaLB Controller. There is a ChannelAddress match if and only if the ChannelAddress recovered from the MediaLB input, MLBSIG, equals the ChannelAddress defined by:
CA[15:0] = { 7'h00, CA[8:1], 1'b0 }.</td></tr>
</table>

27.3.2.12 Channel *n* Status Configuration Register

The Channel *n* Status Configuration Register (CSCR*n*) reflects the status of the given logical channel. The definition of some of the bit fields in the CSCR*n* register vary depending on the selected channel type.

Offset: 0x0044 (CSCR0), 0x0054 (CSCR1), 0x0064 (CSCR2), 0x0074 (CSCR3), 0x0084 (CSCR4), 0x0094 (CSCR5), 0x00A4 (CSCR6), 0x00B4 (CSCR7), 0x00C4 (CSCR8), 0x00D4 (CSCR9), 0x00E4 (CSCR10), 0x00F4 (CSCR11), 0x0104 (CSCR12), 0x0114 (CSCR13), 0x0124 (CSCR14), 0x0134 (CSCR15) Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	BM	BF	0	0	0	0	0	0	0	0	0	0	IVB[1:0]		GIRB/ GB	RDY
W																
Reset	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	PBS	PBD	PBDB	PBPE	0	LFS	HBE	BE	CBS	CBD	CBDB	CBPE
W					w1c	w1c	w1c	w1c		w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 27-13. Channel *n* Status Configuration Register

Table 27-19. Channel *n* Status Configuration Register Field Descriptions

Field	Description
BM	Buffer Empty. When set, the local channel buffer (for channel *n*) is empty. This bit is set and cleared by hardware. At reset, the local channel buffer is empty (BM = 1). 0 Buffer not empty. 1 Buffer empty.
BF	Buffer Full. When set, the local channel buffer (for channel *n*) is full. This bit is set and cleared by hardware. 0 Buffer not full. 1 Buffer full.
IVB[1:0]	Isochronous Valid Bytes. These bits are loaded by hardware with the number of valid bytes in the last packet of a broken Isochronous RX channel. Used in conjunction with CCBCR*n*[BCA], IVB[1:0] can be used by software to determine the final valid byte of the local channel buffer. This field is only valid for logical channels configured for isochronous RX data. 00 Final valid byte = (CCBCR*n*[BCA] – 5). 01 Final valid byte = (CCBCR*n*[BCA] – 4). 10 Final valid byte = (CCBCR*n*[BCA] – 3). 11 Final valid byte = (CCBCR*n*[BCA] – 2).
GIRB	Generate Isochronous Receive Break. When set, this bit causes hardware to terminate the current packet, flush the local channel buffer, clear the RDY bit, and load IVB[1:0]. This bit is set by system software and cleared by hardware. This field is only valid for logical channels configured for isochronous RX data. 0 Do not generate isochronous receive break. 1 Generate isochronous receive break.
GB	Generate Break. When the local channel buffer is configured for TX data, the setting of this bit causes hardware to send the *AsyncBreak* (26h) or *ControlBreak* (36h) command and stop the transfer. When the local channel buffer is configured for RX data, the setting of this bit causes hardware to send the MediaLB RxStatus *ReceiverBreak* (70h) and stop the transfer. This bit is set by system software and cleared by hardware. This bit is only valid for logical channels configured for asynchronous or control data. 0 Do not generate break. 1 Generate break.

Table 27-19. Channel *n* Status Configuration Register Field Descriptions (continued)

Field	Description
RDY	Next Buffer Ready. System software should set this bit when all the registers, data, and program memory variables are setup and ready to transmit or receive data using DMA. For TX data, the system memory buffer should also be filled. For DMA using ping-pong buffering, hardware clears this bit after the buffer begins to be processed. For DMA using circular buffering, software should clear this bit only when buffer processing needs to halted. If RDY is set before processing of the *Current Buffer* is complete, status for the *Current Buffer* is reported using CSCR*n*[11:8] (status bits for the previous buffer) and CSCR*n*[3:0] is not updated. The CSCR*n*[3:0] bits are only updated when the processing for the *Current Buffer* is complete and the RDY bit has not yet been set. 0 Next buffer is not ready in system memory. 1 Next buffer is ready in system memory.
PBS	Previous Buffer Start. When set, this bit indicates the first quadlet of the *Previous Buffer* has been successfully transmitted or received. The setting of this bit generates a maskable channel interrupt to system software. This bit is valid for all channel types. 0 First quadlet of the *Previous Buffer* has not been successfully transmitted or received. 1 First quadlet of the *Previous Buffer* has been successfully transmitted or received.
PBD	Previous Buffer Done. When set, this bit indicates the last quadlet of the *Previous Buffer* has been successfully transmitted or received. The setting of this bit generates a maskable channel interrupt to system software. This bit is valid for all channel types. The *Done* status is always generated when the processing of a buffer has finished, even if a *Break* or *Error* condition was detected during the packet processing. If *Break* or *Error* occurred, the *Done* status bit is set in addition to the *Break* or *Error* status bit. 0 Last quadlet of the *Previous Buffer* has not been successfully transmitted or received. 1 Last quadlet of the *Previous Buffer* has been successfully transmitted or received.
PBDB	Previous Buffer Detect Break. When set, this bit indicates that either a TX channel has detected a receiver break response, *ReceiverBreak* (70h), or an RX channel has detected a transmitter break command, *ControlBreak* (36h) or *AsyncBreak* (26h), while processing the *Previous Buffer*. The setting of this bit generates a maskable channel interrupt to system software. This bit is valid for all channel types. 0 Break response was not detected while processing the *Previous Buffer*. 1 Break response was detected while processing the *Previous Buffer*.
PBPE	Previous Buffer Protocol Error. When set, this bit indicates that either a TX channel has detected an RxStatus of *ReceiverProtocolError* (72h), a RX channel has detected an invalid command for this channel type, or an additional *AsyncStart* (20h) or *ControlStart* (30h) command has been received while in the middle of a packet. The setting of this bit generates a maskable channel interrupt to system software. This bit is valid for all RX channels and valid for only asynchronous and control TX channels. 0 Protocol error was not detected while processing the *Previous Buffer*. 1 Protocol error was detected while processing the *Previous Buffer*.
LFS	Lost Frame Synchronization. When set, this bit indicates that the logical channel has lost synchronization with the MediaLB frame. The setting of this bit generates a maskable channel interrupt to system software. This bit is valid for synchronous channels only. 0 Frame synchronization not lost. 1 Frame synchronization lost.
HBE	Host Bus Error. When set, this bit indicates that an error occurred on the host bus. If the channel is configured for TX, then the bus error occurred during a read access. If the channel is configured for RX, then the bus error occurred during a write access. The setting of this bit generates a non-maskable channel interrupt to system software. This bit is valid for all channel types. 0 Bus error not detected. 1 Bus error detected.

Table 27-19. Channel *n* Status Configuration Register Field Descriptions (continued)

Field	Description
BE	Buffer Error. When set, this bit indicates that either a TX channel has detected a buffer underflow (e.g. attempted to pop data from an empty buffer), or an RX channel has detected a buffer overflow (e.g. attempted to push data onto a full buffer). The setting of this bit generates a maskable channel interrupt to system software. This bit is valid for synchronous RX/TX and isochronous RX (CECR*n*[FCE] = 0) channels only. 0 TX underflow or RX overflow not detected. 1 TX underflow or RX overflow detected.
CBS	Current Buffer Start. When set, this bit indicates the first quadlet of the *Current Buffer* has been successfully transmitted or received. The setting of this bit generates a maskable channel interrupt to system software. This bit is valid for all channel types. 0 First quadlet of the *Current Buffer* has not been successfully transmitted or received. 1 First quadlet of the *Current Buffer* has been successfully transmitted or received.
CBD	Current Buffer Done. When set, this bit indicates the last quadlet of the *Current Buffer* has been successfully transmitted or received. The setting of this bit generates a maskable channel interrupt to system software. This bit is valid for all channel types. The *Done* status is always generated when the processing of a buffer has finished, even if a *Break* or *Error* condition was detected during the packet processing. If *Break* or *Error* occurred, the *Done* status bit is set in addition to the *Break* or *Error* status bit. 0 Last quadlet of the *Current Buffer* has not been successfully transmitted or received. 1 Last quadlet of the *Current Buffer* has been successfully transmitted or received.
CBDB	Current Buffer Detect Break. When set, this bit indicates that either a TX channel has detected a receiver break response, *ReceiverBreak* (70h), or an RX channel has detected a transmitter break command, *ControlBreak* (36h) or *AsyncBreak* (26h), while processing the *Current Buffer*. The setting of this bit generates a maskable channel interrupt to system software. This bit is valid for all channel types. 0 Break response was not detected while processing the *Current Buffer*. 1 Break response was detected while processing the *Current Buffer*.
CBPE	Current Buffer Protocol Error. When set, this bit indicates that either a TX channel has detected an RxStatus of *ReceiverProtocolError* (72h), a RX channel has detected an invalid command for this channel type, or an additional *AsyncStart* (20h) or *ControlStart* (30h) command has been received while in the middle of a packet. The setting of this bit generates a maskable channel interrupt to system software. This bit is valid for all RX channels and valid for only asynchronous and control TX channels. 0 Protocol error was not detected while processing the *Current Buffer*. 1 Protocol error was detected while processing the *Current Buffer*.

NOTE

If the MLB DMA attempts to write to an invalid memory address, then the Host Bus Error bit of Channel N Status Configuration Register (CSCRn[HBE]) is not set as expected. CSCRn[HBE] will be set correctly if MLB DMA attempts to read from an invalid memory address. Therefore, ensure MLB DMA writes are to valid memory addresses.

27.3.2.13 Channel *n* Current Buffer Configuration Register

The Channel *n* Current Buffer Configuration Register (CCBCR*n*) allows system software to monitor the address pointer and buffer length of the *Current Buffer* in system memory for the logical channel. The definitions of the bit fields in the CCBCR*n* register vary depending on the selected channel type.

Offset: 0x0048 (CCBCR0) 0x0058 (CCBCR1) 0x0068 (CCBCR2) 0x0078 (CCBCR3) 0x0088 (CCBCR4) 0x0098 (CCBCR5) 0x00A8 (CCBCR6) 0x00B8 (CCBCR7) 0x00C8 (CCBCR8) 0x00D8 (CCBCR9) 0x00E8 (CCBCR10) 0x00F8 (CCBCR11) 0x0108 (CCBCR12) 0x0118 (CCBCR13) 0x0128 (CCBCR14) 0x0138 (CCBCR15) Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	BCA[15:0]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	BFA[15:0]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 27-14. Channel *n* Current Buffer Configuration Register

Table 27-20. Channel *n* Current Buffer Configuration Register Field Descriptions

Field	Description
BCA [15:0]	Buffer Current Address. The BCA field defines a 16-bit address pointer, which identifies the lower half of the beginning address of the *Current Buffer* in system memory. The BCA[15:2] bits are loaded from CNBCR*n*[BSA[15:2]] when the *Next Buffer* is ready for processing. This *Current Buffer* address pointer should always be quadlet aligned (e.g. BCA[1:0] equals 2'b00). During the processing of the *Current Buffer*, the BCA field marks which quadlet of the buffer is currently being processed. The upper half of the beginning address of the *Current Buffer* is system memory is defined by SBCR[SRBA], ABCA[ARBA], CBCR[CRBA], or IBCR[IRBA] when CECR*n*[TR] is clear; SBCR[STBA], ABCA[ATBA], CBCR[CTBA], or IBCR[ITBA] when CECR*n*[TR] is set, depending on the value of CECR*n*[CT[1:0]].
BFA [15:0]	Buffer Final Address. The BFA field defines a 16-bit address pointer, which identifies the lower half of the ending address of the *Current Buffer* in system memory. The BFA[15:2] bits are loaded from CNBCR*n*[BEA[15:2]] when the *Next Buffer* is read for processing. This *Current Buffer* address pointer, except when associated with isochronous channels, should always be quadlet aligned (e.g. BFA[1:0] equals 2'b00). During the processing of the *Current Buffer*, the point at which the BCA field becomes equal to (or greater than) the BFA field indicates that the processing of the *Current Buffer* ends upon successful completion of the current quadlet (for isochronous and synchronous channels) or upon successful completion of the current packet (for asynchronous and control channels). It is the responsibility of system software to ensure the system memory buffers (for RX asynchronous and control channels) can accommodate overflow in the size of the largest packet supported. Additionally, single-packet buffering can be used by simply programming CNBCR*n*[BSA[15:2]] = CNBCR*n*[BEA[15:2]]. The upper half of the ending address of the *Current Buffer* in system memory is defined by SBCR[SRBA], ABCA[ARBA], CBCR[CRBA], and IBCR[IRBA] when CECR*n*[TR] is clear; SBCR[STBA], ABCA[ATBA], CBCR[CTBA], or IBCR[ITBA] when CECR*n*[TR] is set, depending on the value of CECR*n*[CT[1:0]].

27.3.2.14 Channel *n* Next Buffer Configuration Register

The Channel *n* Next Buffer Configuration Register (CNBCR*n*) allows system software to monitor the address pointer and buffer length of the *Next Buffer* in system memory for the logical channel. The definitions of the bit fields in the CNBCR*n* register vary depending on the selected channel type.

Offset: 0x004C (CNBCR0) 0x005C (CNBCR1) 0x006C (CNBCR2) 0x007C (CNBCR3) 0x008C (CNBCR4) 0x009C (CNBCR5) 0x00AC (CNBCR6) 0x00BC (CNBCR7) 0x00CC (CNBCR8) 0x00DC (CNBCR9) 0x00EC (CNBCR10) 0x00FC (CNBCR11) 0x010C (CNBCR12) 0x011C (CNBCR13) 0x012C (CNBCR14) 0x013C (CNBCR15) Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	BSA[15:0]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	BEA[15:0]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 27-15. Channel *n* Next Buffer Configuration Register

Table 27-21. Channel *n* Next Buffer Configuration Register Field Descriptions

Field	Description
BSA [15:0]	Buffer Start Address. The BSA field defines a 16-bit address pointer, which identifies the lower half of the beginning address of the *Next Buffer* in system memory. Once system software detects CSCR*n*[RDY]has been cleared by hardware (for ping-pong buffering), the beginning address of the *Next Buffer* may be loaded into BSA[15:2]. System software should then set CSCR*n*[RDY]. Once processing of the *Current Buffer* for the logical channel is complete, the BSA[15:2] field is loaded into the CCBCR*n*[BCA[15:2]] field and processing of the next buffer can begin. This *Next Buffer* address pointer must always be quadlet aligned (e.g. BSA[1:0] must be written as 2'b00). The upper half of the beginning address of the *Next Buffer* in system memory is defined by SBCR[SRBA], ABCA[ARBA], CBCR[CRBA], or IBCR[IRBA] when CECR*n*[TR] is clear; SBCR[STBA], ABCA[ATBA], CBCR[CTBA, or IBCR[ITBA] when CECR*n*[TR] is set, depending on the value of CECR*n*[CT[1:0]].
BEA [15:0]	Buffer End Address. The BEA field defines a 16-bit address pointer, which identifies the lower half of the ending address of the *Next Buffer* in system memory. Once system software detects CSCR*n*[RDY] has been cleared by hardware (for ping-pong buffering), the ending address of the *Next Buffer* may be loaded into BEA[15:2]. System software should then set CSCR*n*[RDY]. Once processing of the *Current Buffer* for the logical channel is complete, the BEA[15:2] field is loaded into the CCBCR*n*[BFA[15:2]] field and processing of the next buffer can begin. The BEA[15:2] bits are loaded into CCBCR*n*[BFA[15:2]] when the *Current Buffer* is finished being processed. This *Next Buffer* address pointer, except when associated with isochronous channels, should always be quadlet aligned (e.g. BEA[1:0] defaults to 2'b00). The upper half of the ending address of the *Next Buffer* in system memory is defined by SBCR[SRBA], ABCA[ARBA], CBCR[CRBA], or IBCR[IRBA] when CECR*n*[TR] is clear; SBCR[STBA], ABCA[ATBA], CBCR[CTBA, or IBCR[ITBA] when CECR*n*[TR] is set, depending on the value of CECR*n*[CT[1:0]].

27.3.2.15 Local Channel *n* Buffer Configuration Register

The Local Channel n Buffer Configuration Register (LCBCR*n*) allows software to optimize use of the local buffer RAM for local channel buffering. This register should only be written by software while the logical channel is disabled (e.g. CECR3[CE] clear disables Channel 3; therefore software may write

LCBCR3). Writing to this register while the corresponding logical channel is enabled may result in unexpected behavior.

Offset: 0x0280 (LCBCR0) 0x0284 (LCBCR1) 0x0288 (LCBCR2) 0x028C (LCBCR3) 0x0290 (LCBCR4) 0x0294 (LCBCR5) 0x0298 (LCBCR6) 0x029C (LCBCR7) 0x02A0 (LCBCR8) 0x02A4 (LCBCR9) 0x02A8 (LCBCR10) 0x02AC (LCBCR11) 0x02B0 (LCBCR12) 0x02B4 (LCBCR13) 0x02B8 (LCBCR14) 0x02BC (LCBCR15) Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	BD[8:3]					
W																
Reset	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	BD[2:0]			0	0	0	0	SA[8:0]								
W																
Reset[1]	1	1	1	0	0	0	0	*	*	*	*	*	*	*	*	*

Figure 27-16. Local Channel *n* Buffer Configuration Register

1 Reset value of SA[8:0] for Channel *n* offset = Channel(*n* – 1)offset + Buffer Depth(Channel(*n* – 1))

Table 27-22. Local Channel *n* Buffer Configuration Register Field Descriptions

Field	Description
BD[8:0]	Buffer Depth. This field defines the depth of the local channel buffer in the local buffer RAM in increments of 4 quadlets. At reset, the LCBCH*n*[BD[8:0]] field is loaded with 0x01F, or 128 quadlets. 0x000 – Depth = 4 quadlets. 0x001 – Depth = 8 quadlets. 0x002 – Depth = 12 quadlets. ... 0x1FF – Depth = 2048 quadlets. Value 0x01F (decimal 31) equates to 128 quadlets. The default buffer depth for all channels is 128 quadlets (0x01F).
SA[8:0]	Buffer Start Address. This field defines the starting address of the channel buffer space in the local buffer RAM in increments of 4 quadlets. At reset, the LCBCR*n*[SA[9:0]] field is loaded with the channel number multiplied by 32 (or channel number multiplied by 128 quadlets). 0x000 – RAM Start Address offset = 0 quadlets. 0x001 – RAM Start Address offset = 4 quadlets. 0x002 – RAM Start Address offset = 8 quadlets. ... 0x1FFh – RAM Start Address offset = 2044 quadlets. General: Channel *n* offset = Channel (*n* – 1)offset + BD(Channel (*n* – 1)). Channel 0 = 0 offset. Channel 1 = Channel 0 + BD(16quadlets) = 16quadlets offset. Channel 2 = Channel 1 (16quadlets) + BD(16 quadlets) = 32 quadlets offset. Channel 3 = Channel 2 (32 quadlets) + BD(144 quadlets) = 176 quadlets offset. Channel 4 = Channel 3 (176 quadlets) + BD(144 quadlets) = 320 quadlets offset.

27.4 Functional Description

The MLB Device peripheral is divided into six main components, as illustrated in Figure 27-1.

- MLB Core. Implements the physical layer of the MLB interface. This physical layer performs serial-to-parallel and parallel-to-serial data transformations and MLB frame synchronization.
- Clock and Reset Control
- MLB Link Logic. Implements the link layer functionality of the MLB interface, including:
  - Checking of synchronous, asynchronous, control, and isochronous channel protocol
  - Handling of both RX and TX initiated breaks
  - Generating RX responses to the MLB Core
  - Generating TX commands for the MLB Core
  - Processing of and responding to the system channel commands
  - Detection of MLB bus lock/unlock
  - Recognition and pipe-lining of logical *ChannelAddresses*
- MLB Configuration Logic. Implements the memory space for the Configuration Control Registers and Channel Configuration Registers. These configuration and control registers are used to define

various parameters and control the operation of the MLB Device. The registers are accessed through the peripheral bus.

- MLB Channel Buffer. Implements the interface between the MLB Device and a single-port SRAM. Functionality of the MLB Channel Buffer logic block includes:
  - Buffering of logical channel data for bus latency issues
  - Multiplexing of logical channel data in Big- and Little-Endian mode
  - Implementation of hardware loop-back mode between logical channel 0 (RX) and logical channel 1 (TX)
- MLB Channel Arbiter. Functions as the Host Bus master and is responsible for handling requests from the MLB DMA Controller. Some of the Channel Arbiter functionality includes:
  - Operating as a bus-master for DMA accesses to/from system memory
  - Determining priorities of channel DMA requests
  - Granting requests based on round-robin arbitration
  - Routing data and control information between MLB logical channels and the Host Bus interface, and consolidating channel interrupts

The MLB physical layer interfaces directly to the Media local bus. MediaLB is based on a scalable 3-pin interface (MLBCLK, MLBSIG, MLBDAT) designed to operate at a maximum operating frequency of 1024 Fs (49.152 MHz at 48 kHz). The MediaLB topology supports communication among the MediaLB controller (INIC) and other MediaLB devices. The MediaLB controller interfaces directly with the MOST network. The MediaLB interface consists of the MLBCLK clock line, a bi-directional MLBSIG line for signal information and a bi-directional MLBDAT line for data transfer.

The MediaLB topology supports one network controller connected to one or more devices, where the controller is the interface between the MediaLB devices and the MOST network. The controller also generates the MediaLB clock source MLBCLK, that is synchronized to the MOST network and provides the timing for the entire MediaLB interface. The MLBCLK continues to operate even when the MediaLB controller loses lock with the MOST network.

The MLBSIG line is a multiplexed signal which carries the *ChannelAddress* generated by the MediaLB controller, as well as the *Command* and *RxStatus* bytes from the MediaLB devices. The MediaLB controller incorporates MediaLB device functionality. The *ChannelAddress* indicates which device can transmit and which device or devices receive on a particular logical channel.

The MLBDAT line is driven by the transmitting MediaLB device and is received by all other MediaLB devices including the MediaLB controller. The MLBDAT line carries the actual data (synchronous, asynchronous, isochronous, or control). For synchronous stream data transmission, multiple MediaLB devices can receive the same data, in a broadcast fashion. The transmitting MediaLB device indicates the particular type of data transmitted by sending the appropriate command on the MLBSIG line. The commands supported are determined by the MediaLB device’s link layer.

When receiving signals over the bus, the physical layer first detects the framesync information in the bitstream. The physical layer then captures the control and data information and converts it to a parallel format to be passed to the link layer. When detecting the framesync information in the bitstream the physical layer may also place data on the bus that has been converted from its native parallel format to the

serial format required by the MLB interface. Data is transferred over the MLB in quadlets (32-bit words). The MLB protocol supports as many as 32 quadlets per frame.

27.4.1 Clocking Requirements

The system clock (SYS_CLK) requirements for operation are shown in Table 27-23.

Table 27-23. Minimum MediaLB System Clock Requirements

Fs	MLBCLK	Minimum System Clock Speed
44.1 kHz	256 FS	12 MHz
	512 FS	23 MHz
	1024 FS	46 MHz
48.0 kHz	256 FS	13 MHz
	512 FS	25 MHz
	1024 FS	50 MHz
48.1 kHz	256 FS	13 MHz
	512 FS	25 MHz
	1024 FS	50 MHz

27.4.1.1 Reset

Soft reset of the physical and logical channel blocks is provided via the DDCE[MRS] bit.

Hard reset of the physical and logical channel blocks is enabled via the DCCR[MHRE] bit. When set, reception of the global or device system reset commands resets the physical and link layers.

27.4.2 Interrupts

The MLB module generates 18 different interrupts, which are summarized in Table 27-24. For more information on interrupts, please seeChapter 10, Interrupts and Interrupt Controller (INTC).

Table 27-24. MLB Interrupts

Interrupt Name	PXN20 Interrupt Vector	Interrupt Flag Bits	Interrupt Mask Bits
MLB Channel Interrupt	95	CSCR0[20:31] to CSCR15[20:31]	CECR0[9:15] to CECR15[9:15]
MLB System Interrupt	96	SSCR[25:31]	SMCR[25:31]
MLB Logical Channel 0 Interrupt	97	CSCR0[20:31]	CECR0[9:15]
MLB Logical Channel 1 Interrupt	98	CSCR1[20:31]	CECR1[9:15]
MLB Logical Channel 2 Interrupt	99	CSCR2[20:31]	CECR2[9:15]
MLB Logical Channel 3 Interrupt	100	CSCR3[20:31]	CECR3[9:15]

Table 27-24. MLB Interrupts

Interrupt Name	PXN20 Interrupt Vector	Interrupt Flag Bits	Interrupt Mask Bits
MLB Logical Channel 4 Interrupt	101	CSCR4[20:31]	CECR4[9:15]
MLB Logical Channel 5 Interrupt	102	CSCR5[20:31]	CECR5[9:15]
MLB Logical Channel 6 Interrupt	103	CSCR6[20:31]	CECR6[9:15]
MLB Logical Channel 7 Interrupt	104	CSCR7[20:31]	CECR7[9:15]
MLB Logical Channel 8 Interrupt	105	CSCR8[20:31]	CECR8[9:15]
MLB Logical Channel 9 Interrupt	106	CSCR9[20:31]	CECR9[9:15]
MLB Logical Channel 10 Interrupt	107	CSCR10[20:31]	CECR10[9:15]
MLB Logical Channel 11 Interrupt	108	CSCR11[20:31]	CECR11[9:15]
MLB Logical Channel 12 Interrupt	109	CSCR12[20:31]	CECR12[9:15]
MLB Logical Channel 13 Interrupt	110	CSCR13[20:31]	CECR13[9:15]
MLB Logical Channel 14 Interrupt	111	CSCR14[20:31]	CECR14[9:15]
MLB Logical Channel 15 Interrupt	112	CSCR15[20:31]	CECR15[9:15]

27.4.3 System Memory Buffers

System software must define system memory buffers for each hardware channel, using the CCBCR*n* and CNBCR*n* registers. Each system memory buffer can occupy as much as 64 KB and must be aligned on a 64 KB boundary; however, the system memory buffers are not required to be contiguous with each other.

Each of the system memory buffers can be configured for either multi-packet or single-packet buffering. Multi-packet buffering allows the system to reduce the interrupt load at the expense of larger system memory buffers. Single-packet buffering allows system memory buffer size to be reduced at the expense of increasing the interrupt rate.

System memory must accommodate situations in which the end of the buffer does not coincide with the end of the current packet (e.g. asynchronous and control RX packets). This requires the system memory buffers to allow overflow by the worst-case packet length.

System memory buffers are referred to as *Previous Buffer*, *Current Buffer*, and *Next Buffer*. The *Current Buffer* is the system memory buffer the DMA Controller is currently processing and is defined by the CCBCR*n* register. The status of the *Current Buffer* is reflected in CSCR*n*[STS[3:0]]. The *Previous Buffer* is the system memory buffer the DMA Controller completed processing prior to the *Current Buffer*. The status of the *Previous Buffer* is reflected in CSCR*n*[STS[11:8]]. The *Next Buffer* is the system memory buffer the DMA Controller begins processing after the *Current Buffer*. The *Next Buffer* is defined by the CNBCR*n* register.

For Isochronous RX channels, the DMA Controller aligns incoming packets on a packet boundary in system memory, dependent on the setting of CECR*n*[IPL[7:0]] and the arrival of the *IsoSyncByte* command.

27.4.4 Local Channel Buffer RAM

A single-port RAM is used to implement the memory space for local channel buffering. The size of the RAM is 2k x 36-bits (1 quadlet of data; 4-bit tag). The initial start address and depth values for the logical channels buffered in the RAM are controlled via the LCBCR*n*. After reset, the initial settings can be overwritten by software.

See Figure 27-17 for more information on using the LCBCR*n* to configure the local RAM buffer.

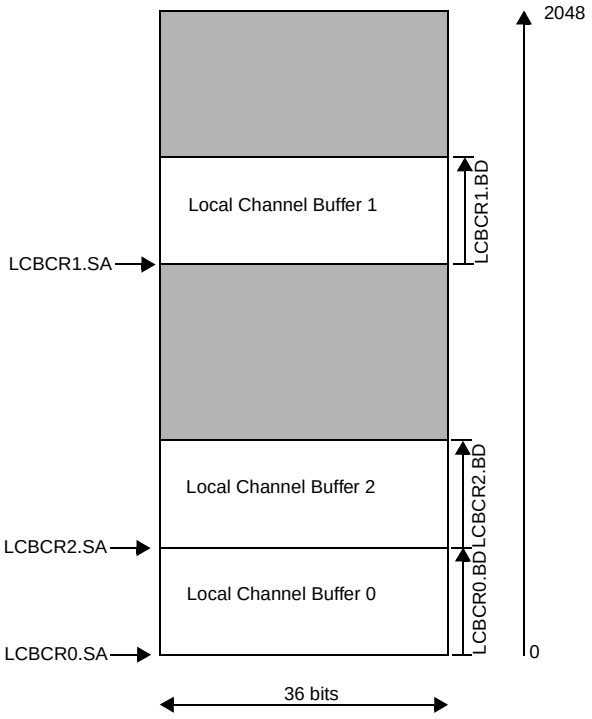


Figure 27-17. Programming Example for LCBCR*n*

27.4.4.1 Local Buffer Start Address

The initial buffer start address of each local channel buffer is defined by the default (after reset) start address of each local channel buffer (LCBCR*n*[SA[8:0]]). The start address is the location of the beginning of the buffer in the local buffer RAM. Software may change the start address of each local channel buffer after reset by writing to LCBCR*n*[SA[8:0]] directly.

27.4.4.2 Local Channel Buffer Depth

The initial buffer depth of each local channel buffer is configured by the default (after reset) depth of each local channel buffer (LCBCR*n*[BD[8:0]]) in quadlets. The buffer depth should be set based on the worst-case DMA interface read/write latency. Software may change the depth of each local channel buffer after reset by writing to LCBCR*n*[BD[8:0]] directly.

27.4.5 Channel Arbiter

The MLB Device includes a DMA Controller with a Host Bus that can access system memory. The channel arbiter logic of the MLB Device, arbitrates requests between the different logical channel requests. Some of the functions of the channel arbiter include:

- determining priorities of channel requests
- granting high priority requests based on round-robin arbitration of current high priority requests
- granting low priority requests based on round-robin arbitration of current low priority requests, and
- routing data and control information between channels and the Host Bus.

27.4.5.1 Round Robin Arbitration

The MLB channel arbiter uses *round-robin arbitration* to determine which logical channel is granted access to the Host Bus. An example of the *round-robin arbitration* method is provided in Figure 27-18.

Round-Robin Arbitration: 4 Channel Example

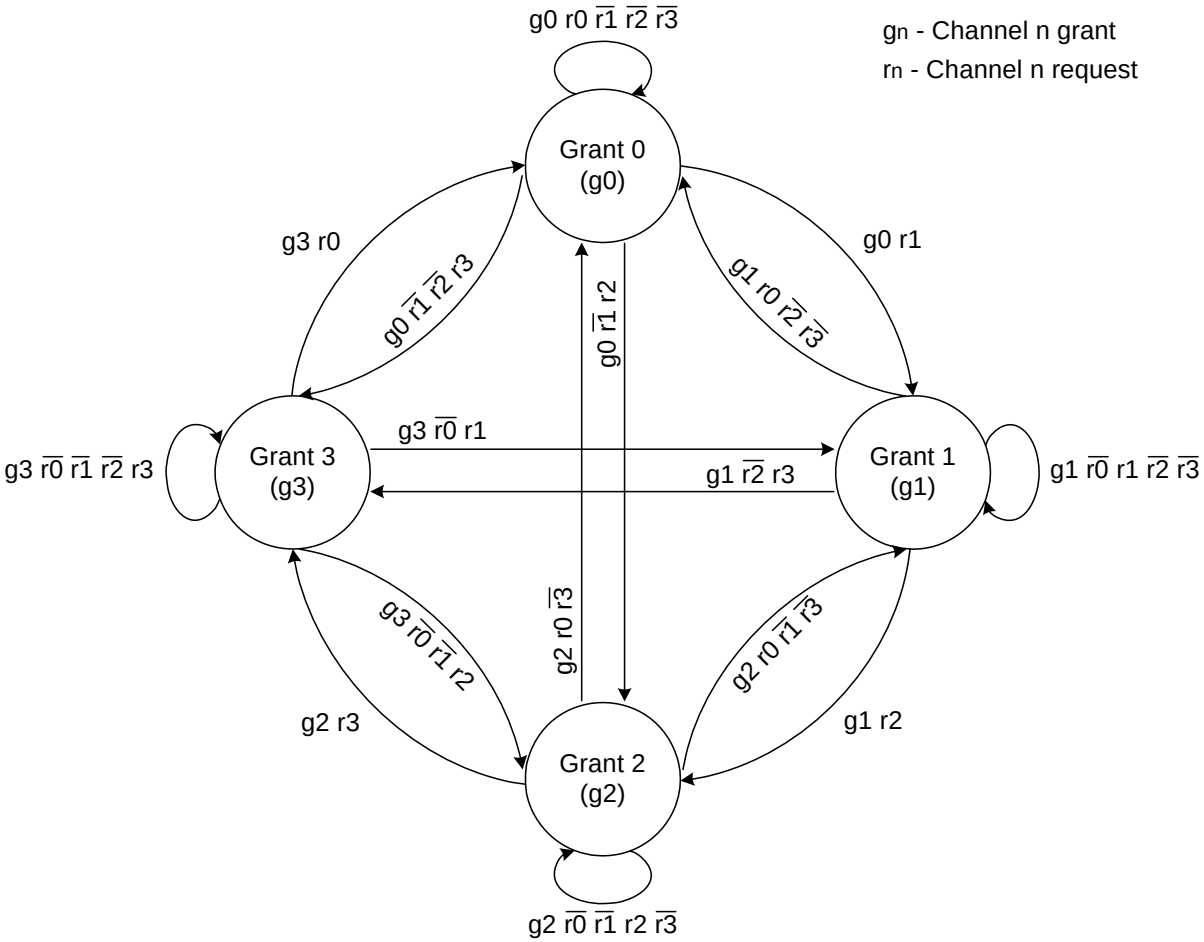


Figure 27-18. Round-Robin Arbitration Example

27.4.6 DMA Controller (Ping-Pong Buffering)

Using the MLB DMA Controller with ping-pong buffering dictates a particular method used for transferring data between hardware channels and system memory. When the MLB hardware channels are configured in this mode, the CCBCR*n* and CNBCR*n* registers are used to configure and monitor the system memory *Current Buffer* and *Next Buffer,* respectively.

Channels use ping-pong buffering when CECR*n*[MDS[1:0]] = 00. The *Current Buffer* and *Next Buffer* are independent system memory buffers, which allow hardware to support the ping-pong buffering. Each is addressed using two 16-bit address pointers, as follows:

- Buffer Start Address (CNBCR*n*[BSA]) – defines the beginning address of the *Next Buffer* in system memory
- Buffer End Address (CNBCR*n*[BEA]) – determines the end of the *Next Buffer* in system memory
- Buffer Current Address (CCBCR*n*[BCA]) – defines the beginning of the *Current Buffer* in system memory
- Buffer Final Address (CCBCR*n*[BFA]) – defines the end of the *Current Buffer* in system memory

27.4.6.1 Asynchronous and Control Packet Handling

The *Current Buffer* and *Next Buffer* can be configured for either multi-packet or single-packet buffering, when receiving and transmitting asynchronous and control packet data. Multi-packet buffering allows the system to reduce the interrupt load at the expense of larger system memory buffers. Single-packet buffering allows system memory buffer size to be reduced at the expense of increasing the interrupt rate.

An example of multi-packet buffering for asynchronous and control channels is provided in Figure 27-19.

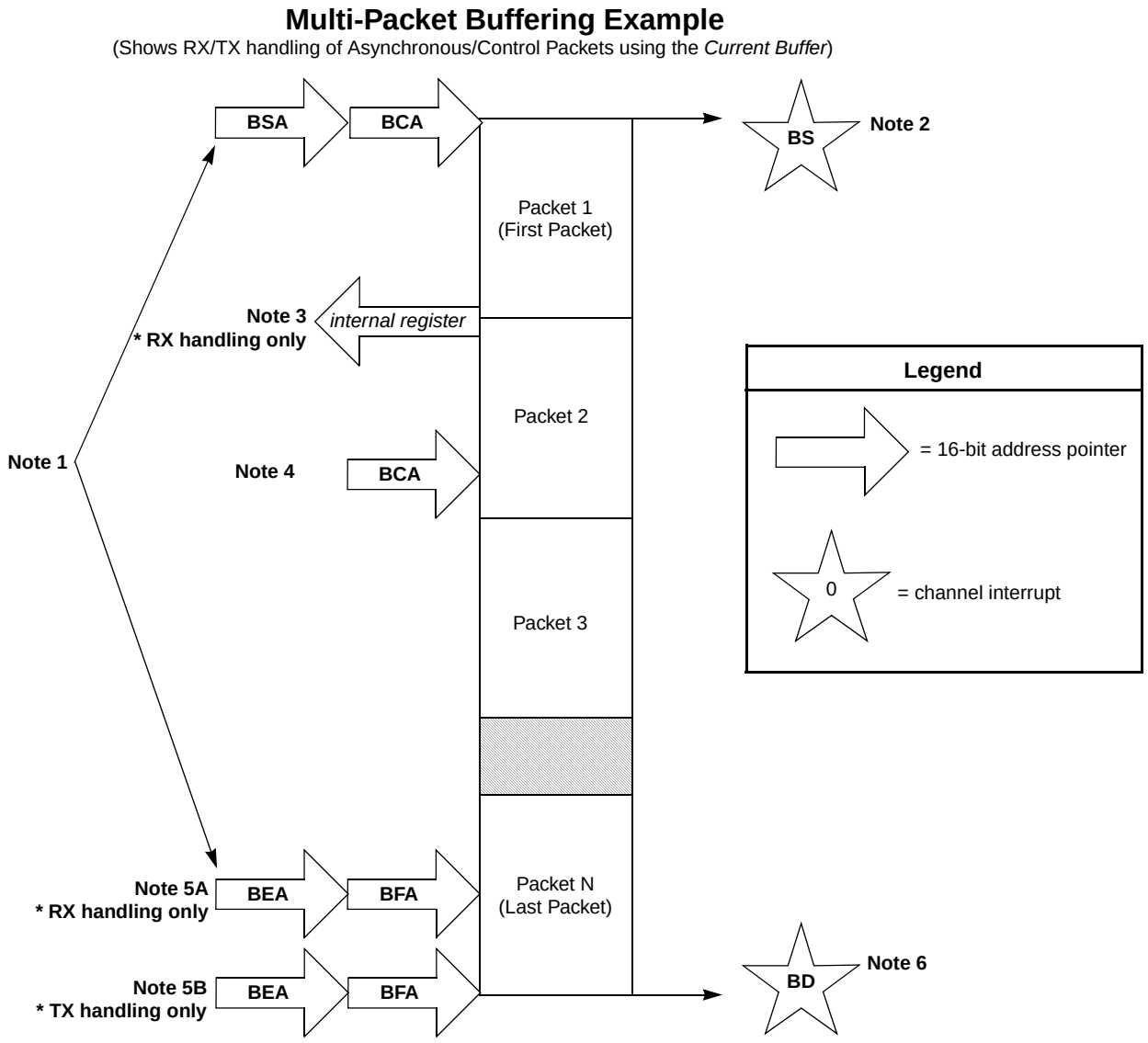


Figure 27-19. Asynchronous/Control Packet Buffering Example

27.4.6.1.1 Packet Reception

When multi-packet buffering is used for receiving asynchronous or control data packets, buffer processing should be handled in the following manner:

- At the start of buffer processing, the beginning of the *Next Buffer* becomes the beginning of the *Current Buffer*, as CNBCR*n*[BSA] is loaded into CCBCR*n*[BCA]. Additionally, the end of the *Next Buffe*r becomes the end of the *Current Buffer*, as CNBCR*n*[BEA] is loaded into CCBCR*n*[BFA] (See Note 1 in Figure 27-19).
- A *Buffer Start* interrupt is generated (CSCR*n*[STS[3]] set), which informs software that hardware has updated CCBCR*n*, cleared the local channel CSCR*n*[RDY] bit, and is available to accept the next buffer. Software may then prepare the *Next Buffer* by writing CNBCR*n*[BSA], CNBCR*n*[BEA], and CSCR*n*[RDY]. (See Note 2 in Figure 27-19).

- The CCBCR*n*[BCA] field is loaded into an internal hardware register (not visible to system software) at the start of each incoming asynchronous or control RX packet. If the packet is later aborted (caused by *AsyncBreak*, *ControlBreak*, *ReceiverBreak*, or *ReceiverProtocolError*), CCBCR*n*[BCA] is restored with the address pointer in the internal hardware register. The next packet then overwrites the aborted RX packet, as aborted RX packets are not stored in system memory. (See Note 3 in Figure 27-19).
- During the processing of the *Current Buffer*, CCBCR*n*[BCA] continues to mark which quadlet of the asynchronous or control RX packet is currently being processed. (See Note 4 in Figure 27-19).
- Software is unable to predict the buffer length for asynchronous and control RX channels, since the length of each RX packet is defined by the packet header (PML) and extracted by hardware as the packet is received. As a result, there is a possibility that the last packet in the *Current Buffer* may extend beyond CCBCR*n*[BFA]. System memory must accommodate this by allowing the buffers to overflow by the worst-case packet length. (See Note 5A in Figure 27-19).
- A *Buffer Done* interrupt is generated (CSCR*n*[STS[2]] set) when the last quadlet from the last packet (in the *Current Buffer*) has been successfully received. Software may then begin processing the buffer. (See Note 6 in Figure 27-19).

NOTE

When the DMA Controller encounters an asynchronous or control packet that is broken (or has an error), CCBCR*n*[BCA] is reloaded with the start address of the *last* packet and the broken packet is overwritten. This mechanism ensures that system software can always calculate the address of the *next* packet start address within system memory.

Single-packet buffering of asynchronous and control RX packets should be handled in the same manner described for multi-packet buffering, with the exception that the beginning and end address of the *Next Buffer* should be set to the same address (e.g. CNBCR*n*[BSA] = CNBCR*n*[BEA]).

27.4.6.1.2 Packet Transmission

When multi-packet buffering is used for transmitting asynchronous or control data packets, buffer processing should be handled in the following manner:

- At the start of buffer processing, the beginning of the *Next Buffer* becomes the beginning of the *Current Buffer*, as CNBCR*n*[BSA] is loaded into CCBCR*n*[BCA]. Additionally, the end of the *Next Buffer* becomes the end of the *Current Buffer*, as CNBCR*n*[BEA] is loaded into CCBCR*n*[BFA]. (See Note 1 in Figure 27-19).
- A *Buffer Start* interrupt is generated (CSCR*n*[STS[3]] set), which informs software that hardware has updated CCBCR*n*, cleared the local channel CSCR*n*[RDY] bit, and is available to accept the next buffer. Software may then prepare the *Next Buffer* by writing: CNBCR*n*[BSA], CNBCR*n*[BEA], and CSCR*n*[RDY]. (See Note 2 in Figure 27-19).
- During the processing of the *Current Buffer*, CCBCR*n*[BCA] continues to mark which quadlet of the asynchronous or control TX packet is currently being processed. (See Note 4 in Figure 27-19).
- System software can determine the exact buffer length for TX channels. As a result, the last packet in the *Current Buffer* should coincide with CCBCR*n*[BFA]. (See Note 5B in Figure 27-19).

- A *Buffer Done* interrupt is generated (CSCR*n*[STS[2]] set) when the last quadlet from the last packet (in the *Current Buffer*) has been successfully transmitted. (See Note 6 in Figure 27-19).

Single-packet buffering of asynchronous and control TX packets should be handled in the same manner described for multi-packet buffering.

27.4.6.2 Isochronous and Synchronous Data Handling

Reception and transmission of isochronous and synchronous data should be handled in the following manner:

- At the start of buffer processing, the beginning of the *Next Buffer* becomes the beginning of the *Current Buffer*, as CNBCR*n*[BSA] is loaded into CCBCR*n*[BCA]. Additionally, the end of the *Next Buffer* becomes the end of the *Current Buffer*, as CNBCR*n*[BEA] is loaded into CCBCR*n*.[BFA]. (See Note 1 in Figure 27-20).
- A *Buffer Start* interrupt is generated (CSCR*n*[STS[3]] set), which informs software that hardware has updated CCBCR*n*, cleared the local channel CSCR*n*[RDY] bit, and is available to accept the next buffer. Software may then prepare the *Next Buffer* by writing CNBCR*n*[BSA], CNBCR*n*[BEA], and CSCR*n*[RDY]. (See Note 2 in Figure 27-20).
- During the processing of the *Current Buffer*, CCBCR*n*[BCA] continues to mark which quadlet of the isochronous or synchronous data is currently being processed. (See Note 3 in Figure 27-20).
- A *Buffer Done* interrupt is generated (CSCR*n*[STS[2]] set) when the last quadlet in the *Current Buffer* has been successfully transmitted/received. (See Note 4 in Figure 27-20).

An example of buffer processing for isochronous and synchronous channels is provided in Figure 27-20.

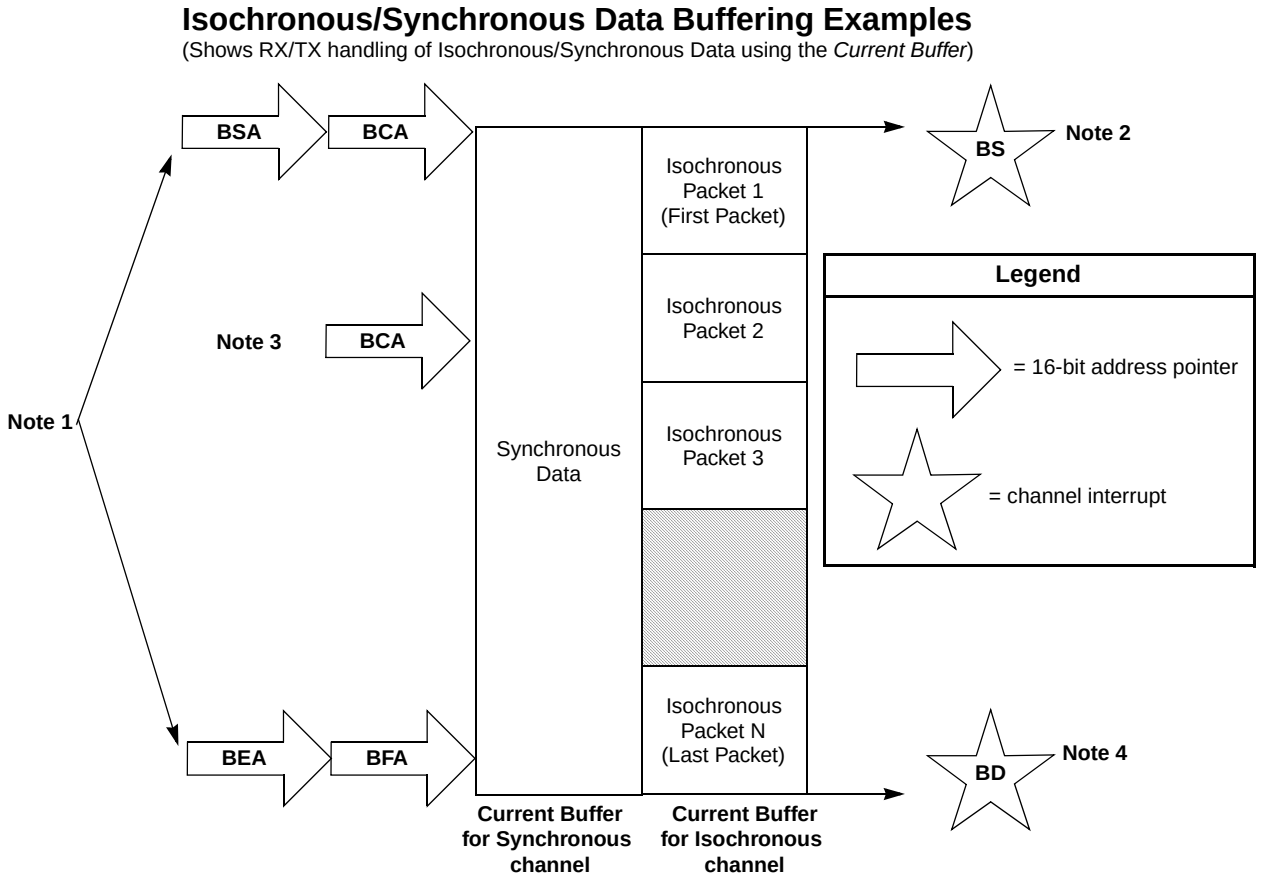


Figure 27-20. Isochronous/Synchronous Data Buffering Examples

For reception or transmission of isochronous data, single-packet and multi-packet buffering is handled in the same manner. Since isochronous channels have a fixed packet length (determined by CECR*n*[IPL]), software should set the system memory buffer length as an even multiple of CECR*n*[IPL] for multi-packet buffering and equal to CECR*n*[IPL] for single-packet buffering. It is assumed that all isochronous packets in the system are of the same length, with the minimum supported length being 5 bytes.

For reception or transmission of synchronous data, the concept of multi-packet or single-packet buffering is not applicable since synchronous data has no packet format. As a result, CCBCR*n*[BFA] always indicates the end address of the *Current Buffer* for synchronous channels.

27.4.7 DMA Controller (Circular Buffering)

Logical channels can be programmed to operate using circular buffering by programming CECR*n*[MDS[1:0]] = 01. It is recommended that circular buffering be used with synchronous channels only (CECR*n*[CT[1:0]] = 00). Logical channels configured for transmitting or receiving other types of data (e.g. asynchronous, control, or isochronous) should not use circular buffering.

In contrast ping-pong buffering, this mode effectively uses a single, circular system memory buffer to process channel data. Software must program the beginning and ending address of the circular buffer in the CNBCR*n*[BSA] and CNBCR*n*[BEA] fields. For proper operation, software must not change the addresses in CNBCR*n*[BSA] and CNBCR*n*[BEA] once buffer processing has started.

While processing the circular buffer, the CSCR*n*[RDY] bit is not automatically cleared by hardware, as it is with ping-pong buffering. For circular buffer, the CSCR*n*[RDY] bit can only be cleared by software through the peripheral bus interface. Once CNBCR*n*[BSA] and CNBCR*n*[BEA] are initially loaded, software should set the CSCR*n*[RDY] bit to initiate buffer processing. This bit may be cleared by software, as needed, to halt the buffer processing.

System design must ensure synchronous data is loaded into the circular buffer at the same rate at which it is unloaded.

An example of the circular buffering for synchronous channels is provided in Figure 27-21.

Synchronous Data Circular Buffering Example

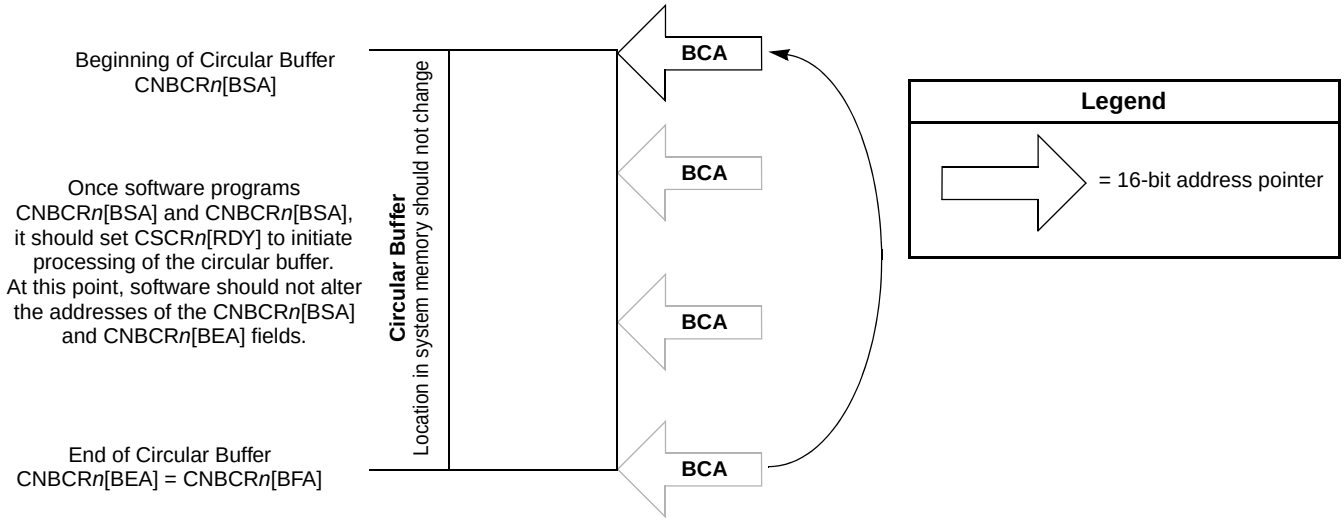


Figure 27-21. Circular Buffering of Synchronous Data

Synchronous data using circular buffering should be handled in the following manner:

- Before buffer processing can begin, software must define the beginning address (CNBCR*n*[BSA]) and ending address of the circular buffer (CNBCR*n*[BEA]). Once the circular buffer beginning and ending addresses are defined, software must set the CSCR*n*[RDY] bit to initiate buffer processing.
- At the start of buffer processing, the beginning address of the circular buffer (CNBCR*n*[BSA]) is loaded into CCBCR*n*[BCA] CCBCR*n*[BCA]. Additionally, the ending address of the circular buffer (CNBCR*n*[BEA]) is loaded into CCBCR*n*[BFA].
- During the processing of the circular buffer, CCBCR*n*[BCA] is updated to indicate which quadlet of the synchronous data is currently being processed.
- Once the end of the buffer is reached and CCBCR*n*[BCA] = CCBCR*n*[BFA], the CCBCR*n*[BCA] field is reloaded to point to the beginning address of the circular buffer (CNBCR*n*[BSA]).
- The CSCR*n*[RDY] bit remains set during the processing of the circular buffer. Software may clear this bit, as needed, to halt buffer processing.

27.4.8 Streaming Channel Frame Synchronization

Certain types of streaming applications require data to be synchronous with the MLB frame, including: stereo, 5.1 audio, and Generic Synchronous Packet Format (GSPF) DTCP. The MLB *Streaming Channel Frame Synchronization* feature provides this option.

For example, 24-bit stereo channels require two MLB physical channels (PC) to transmit left (0xLLLLLn) and right (0xRRRRRn) speaker data. Assuming the MLB Controller allocates Physical Channel 1 (PC1) and Physical Channel 2 (PC2) to this stereo channel, the data would be synchronized to the MLB frame as shown in Table 27-25.

Table 27-25. Example of 24-bit Stereo Data Synchronous to 256 Fs MediaLB frame

Frame	PC = 0	PC = 1	PC = 2	PC = 3	PC = 4	PC = 5	PC = 6	PC = 7
n = 0		0xLLLL_LLRR	0xRRRR_xxxx					
n = 1		0xLLLL_LLRR	0xRRRR_xxxx					
n = 2		0xLLLL_LLRR	0xRRRR_xxxx					
n = 3		0xLLLL_LLRR	0xRRRR_xxxx					

Without frame synchronization, the MLB may begin transmitting or receiving data that is not aligned with the MLB frame. Misalignment, as depicted in Table 27-26, may result in data corruption.

Table 27-26. Example of 24-bit Stereo Data Asynchronous to 256 Fs MediaLB frame

Frame	PC = 0	PC = 1	PC = 2	PC = 3	PC = 4	PC = 5	PC = 6	PC = 7
n = 0			0xLLLL_LLRR					
n = 1		0xRRRR_xxxx	0xLLLL_LLRR					
n = 2		0xRRRR_xxxx	0xLLLL_LLRR					
n = 3		0xRRRR_xxxx	0xLLLL_LLRR					

The MLB supports *Streaming Channel Frame Synchronization* as a programmable option for each logical channel configured for synchronous dataflow. System software can enable the frame synchronization feature for a synchronous logical channel by setting CECHR*n*[FSE]. When enabled, the synchronous logical channel begins transmitting and receiving data only at a MLB frame boundary.

When the loss of MLB frame synchronization occurs, the MLB detects it and optionally notifies system software via a maskable channel interrupt. In order to use this option, system software must:

- program CECR*n*[FSPC[4:0]] with the expected number of physical channels per frame for the logical channel, and
- unmask the CSCR*n*[STS[6]] bit by setting CECR*n*[MLFS] to 0.

A channel interrupt is generated when the actual number of physical channels detected during a MLB frame does not match the expected value. An additional channel interrupt is generated if the local channel buffer overflows (for RX channels) or underflows (for TX channels).

Additionally, software may instruct the MLB to automatically disable a logical channel when MLB frame synchronization is lost. To enable this feature, software must set CSCLR*n*[FSCD], which causes hardware to automatically clear the Channel Enable bit (CECHR*n*[CE]) when synchronization is lost.

Frame synchronization is not supported for asynchronous, control, or isochronous channels.

27.4.9 Loop Back Test Mode

In order to facilitate silicon debug of the MLB Device, hardware supports the *Loop-Back Test Mode*. This mode allows testing of the MLB pads, physical layer, link layer, channel protocol, and local channel buffer. When the DCCR[LBM] bit is set, a data path is enabled which allows RX data from Channel 0 to be sent out as TX data on Channel 1.

Figure 27-22 illustrates the *Loop-Back Test Mode* data path.

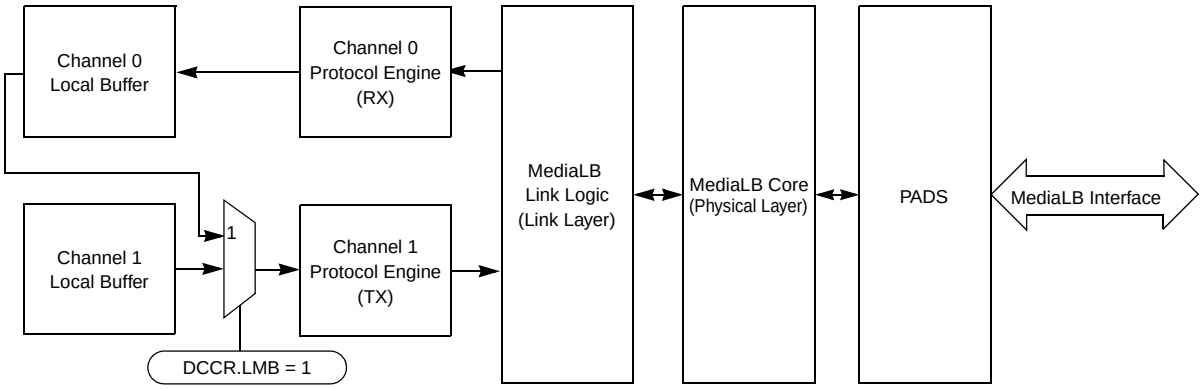


Figure 27-22. Loop-Back Test Mode Data Path

For *Loop-Back Test Mode* operation, software must perform the following steps:

- Set the logical *ChannelAddresses* for Channel 0 and 1. (They cannot be the same address.)
- Enable Channel 0 for receiving synchronous, asynchronous, control, or isochronous data.
- Enable Channel 1 for transmitting the same channel data type as Channel 0.
- Set the Loop-Back Mode bit (DCCR[LBM]).

Restrictions on the *Loop-Back Test Mode* are as follows:

- No protocol errors or breaks are allowed on either the RX or TX channel.
- Little-Endian mode must be disabled (DCCR[MLE] clear).
- Isochronous packet lengths must be quadlet multiples.
- Next Buffer Ready bits for Channels 0 and 1 must remain clear (CSCR0[RDY] = CSCR1[RDY] = 0)

27.5 Initialization Information

The flowcharts in the following pages detail the intended software flow for the MLB Device peripheral. These flowcharts are provided as examples only and are not intended as a source for firmware. Other valid software flows are possible.

27.5.1 Main Loop

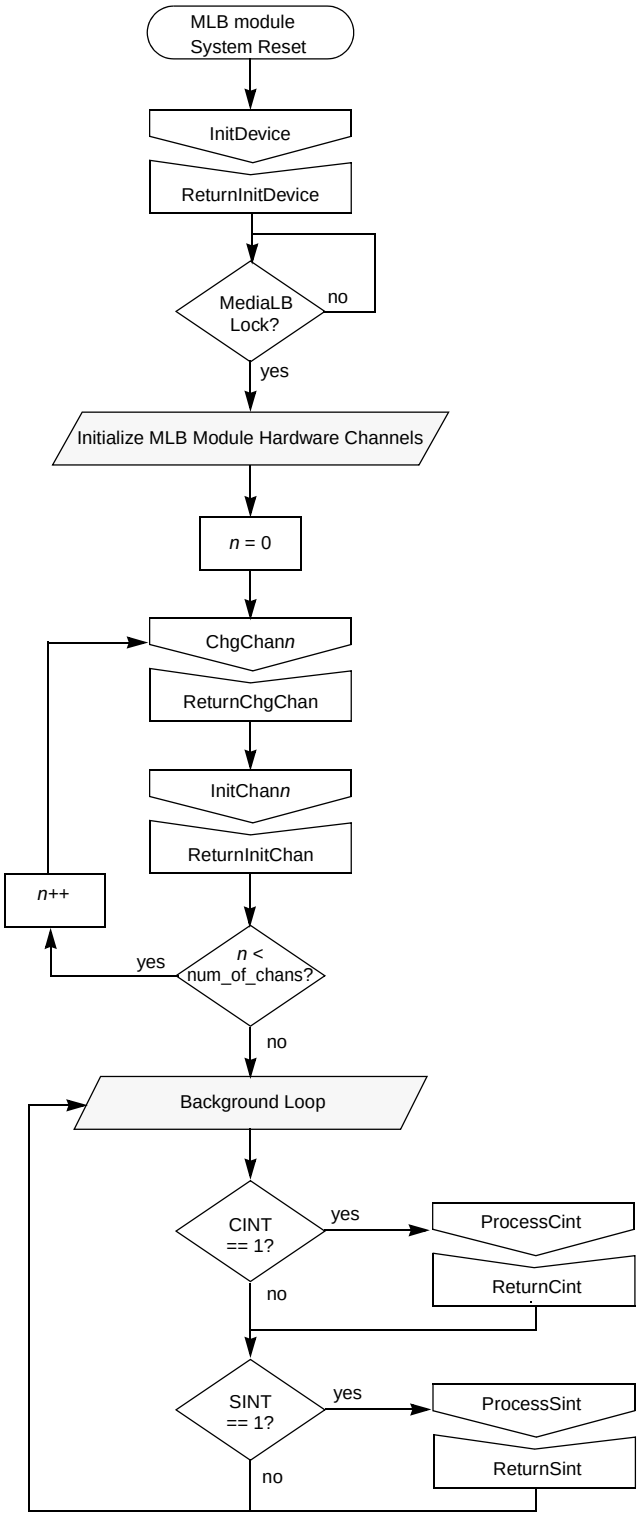


Figure 27-23. Main Loop

27.5.2 Initialize Device

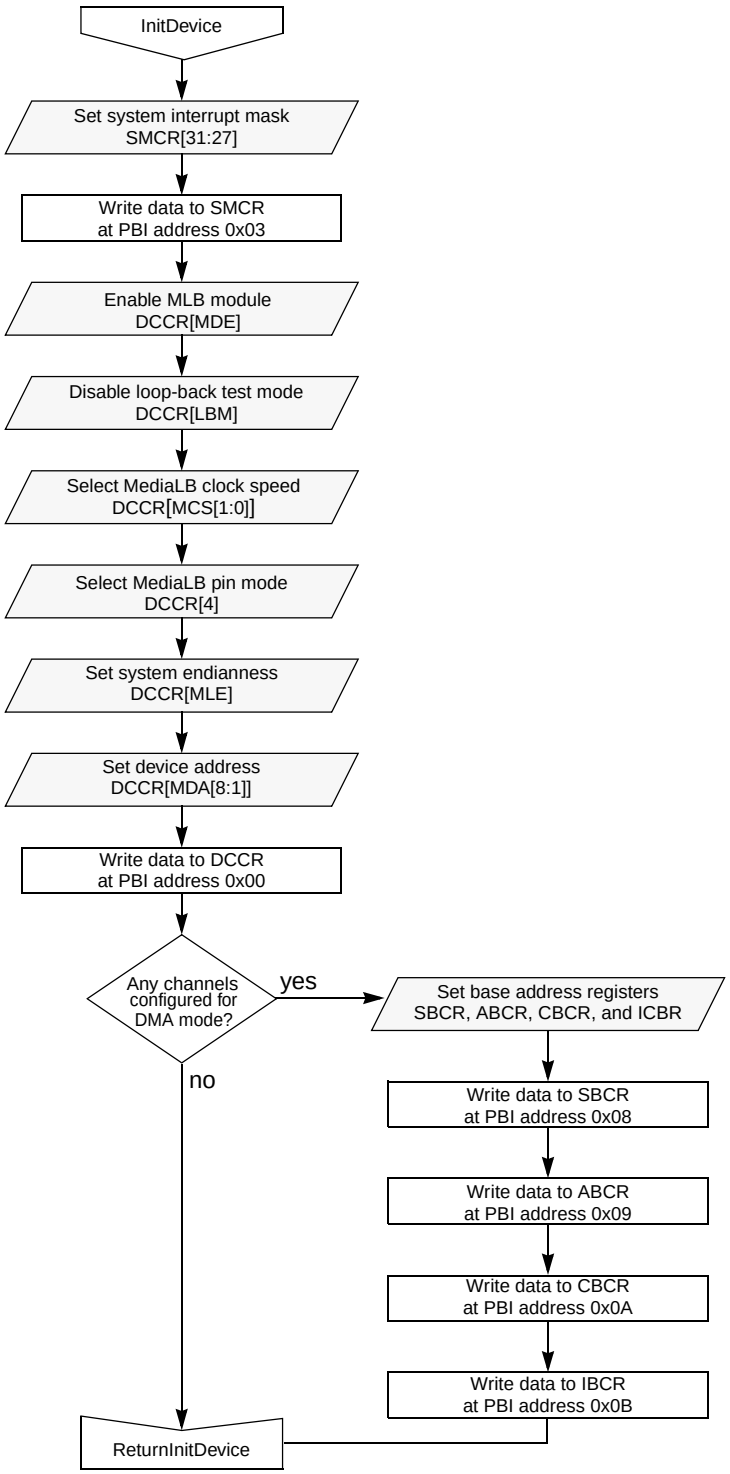


Figure 27-24. Initialize Device

27.5.3 Initialize Channel

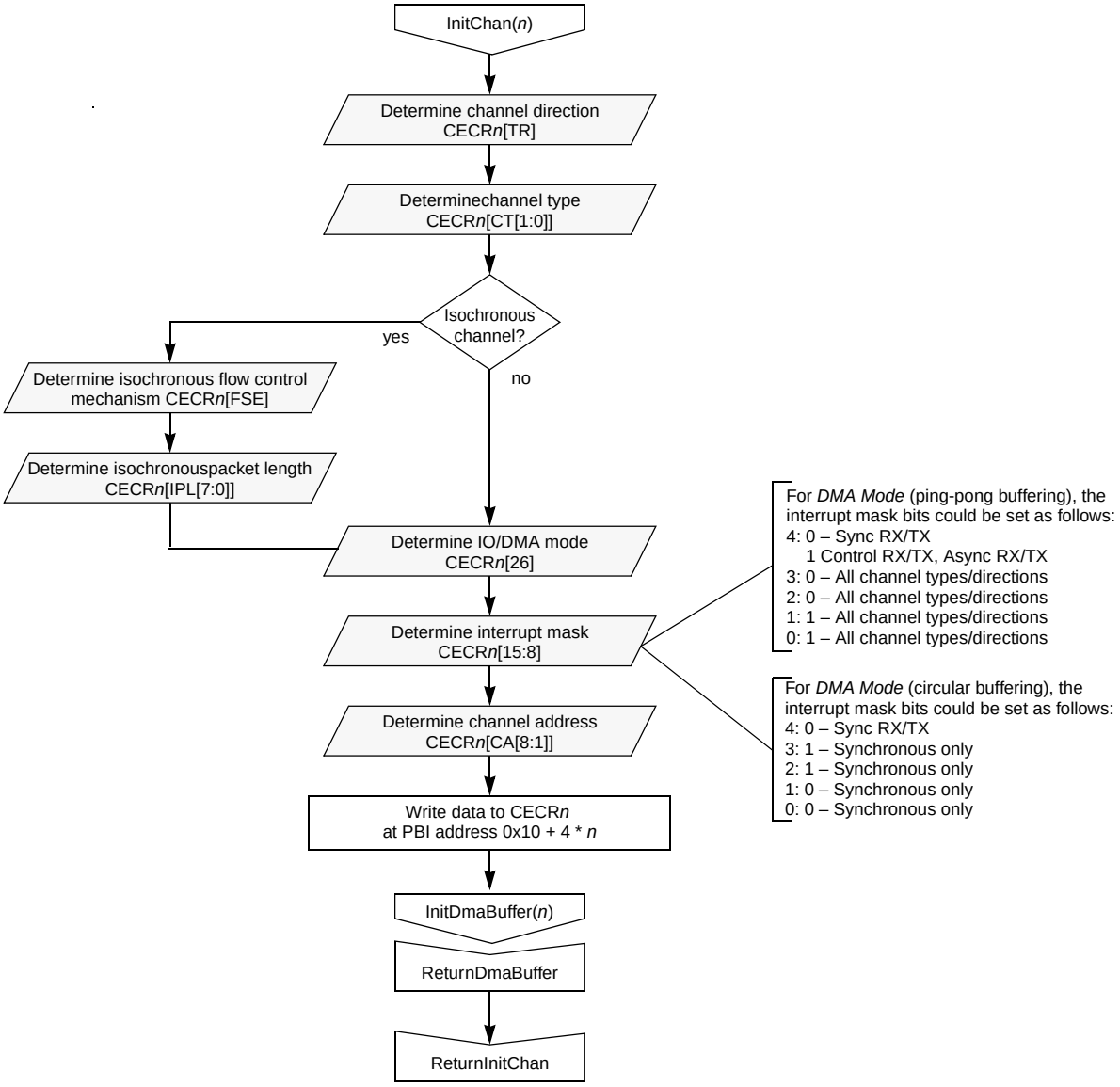


Figure 27-25. Initialize Channel

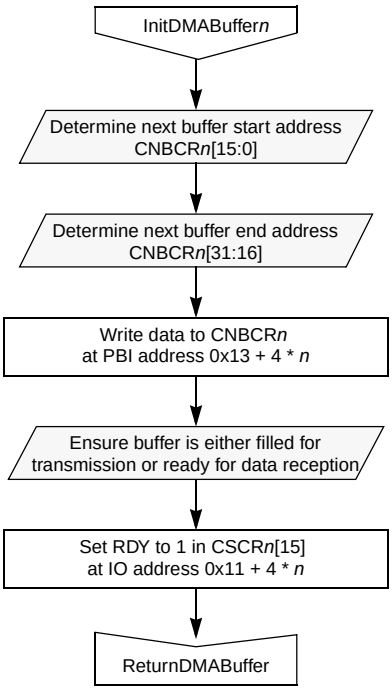


Figure 27-26. Initialize DMA Buffer

27.5.4 Channel Interrupts

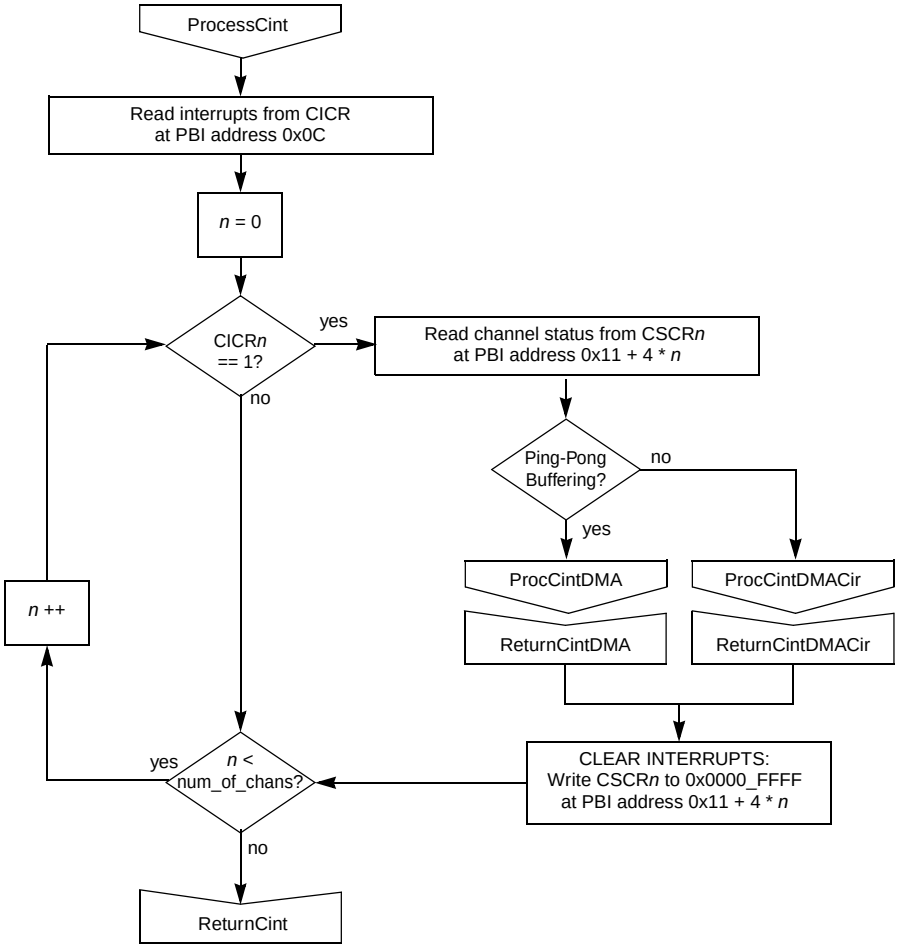


Figure 27-27. Channel Interrupt

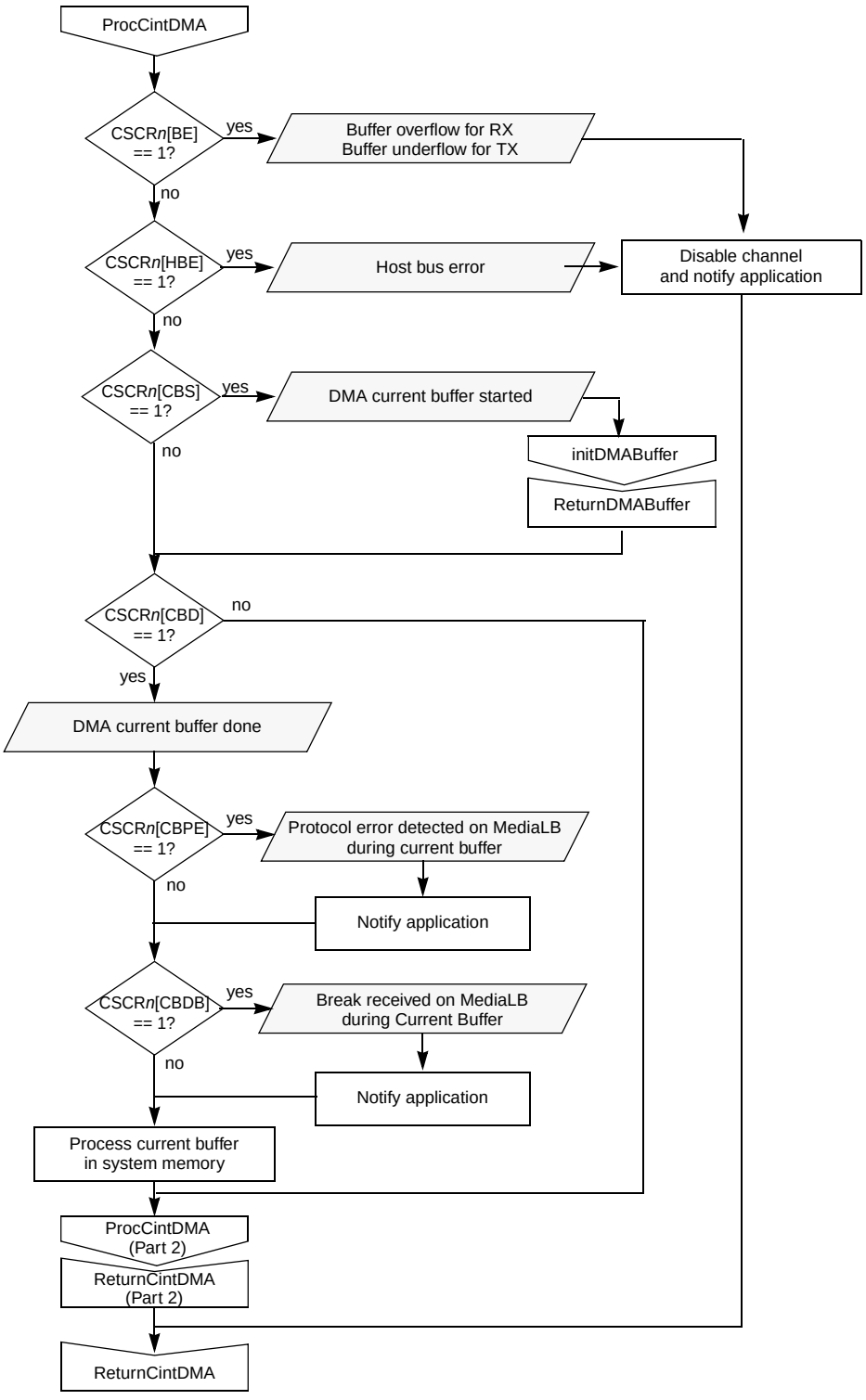


Figure 27-28. DMA Ping-Pong Buffer Channel Interrupt (part 1)

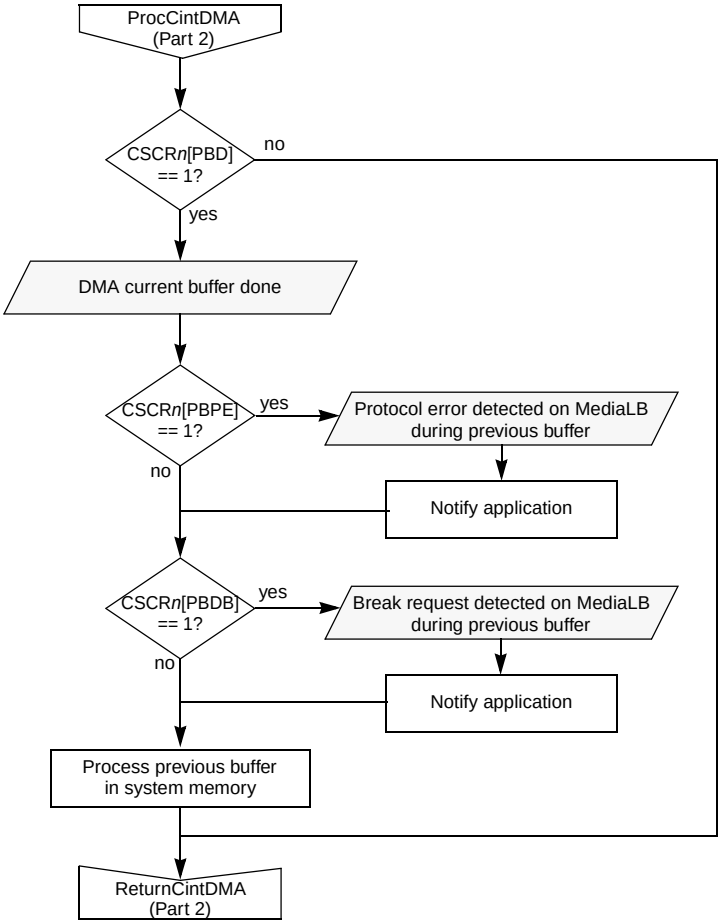


Figure 27-29. DMA Ping-Pong Buffer Channel Interrupt (part 2)

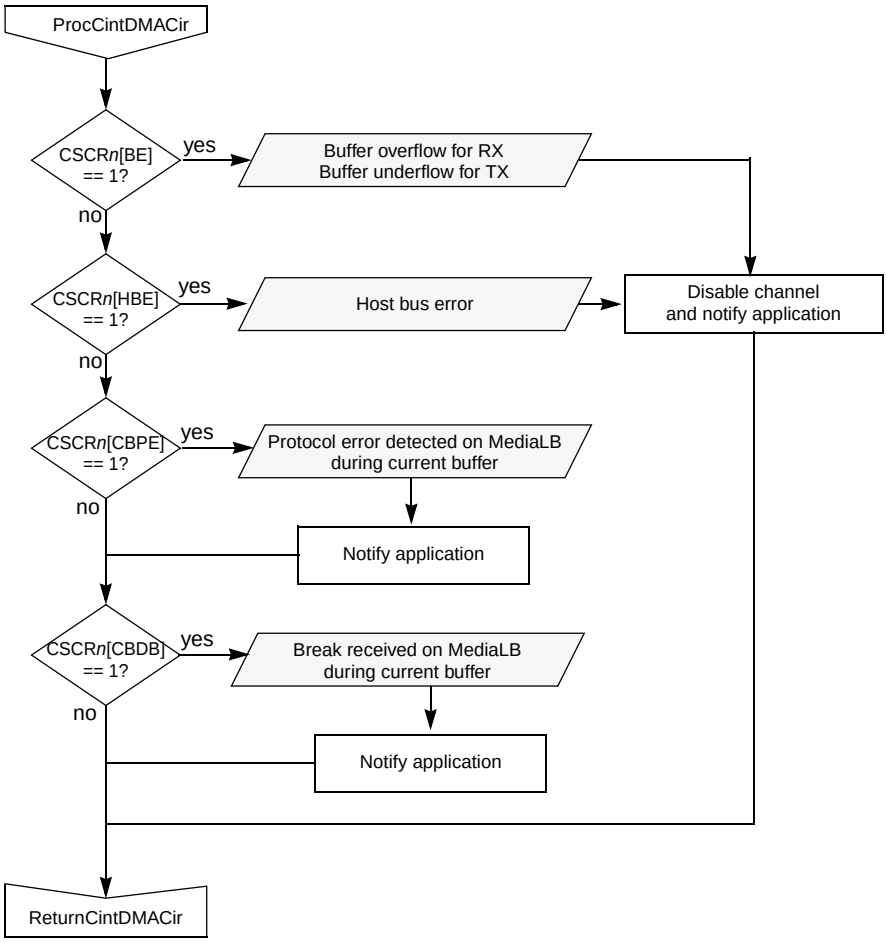


Figure 27-30. DMA Circular Buffer Channel Interrupt

27.5.5 System Interrupts

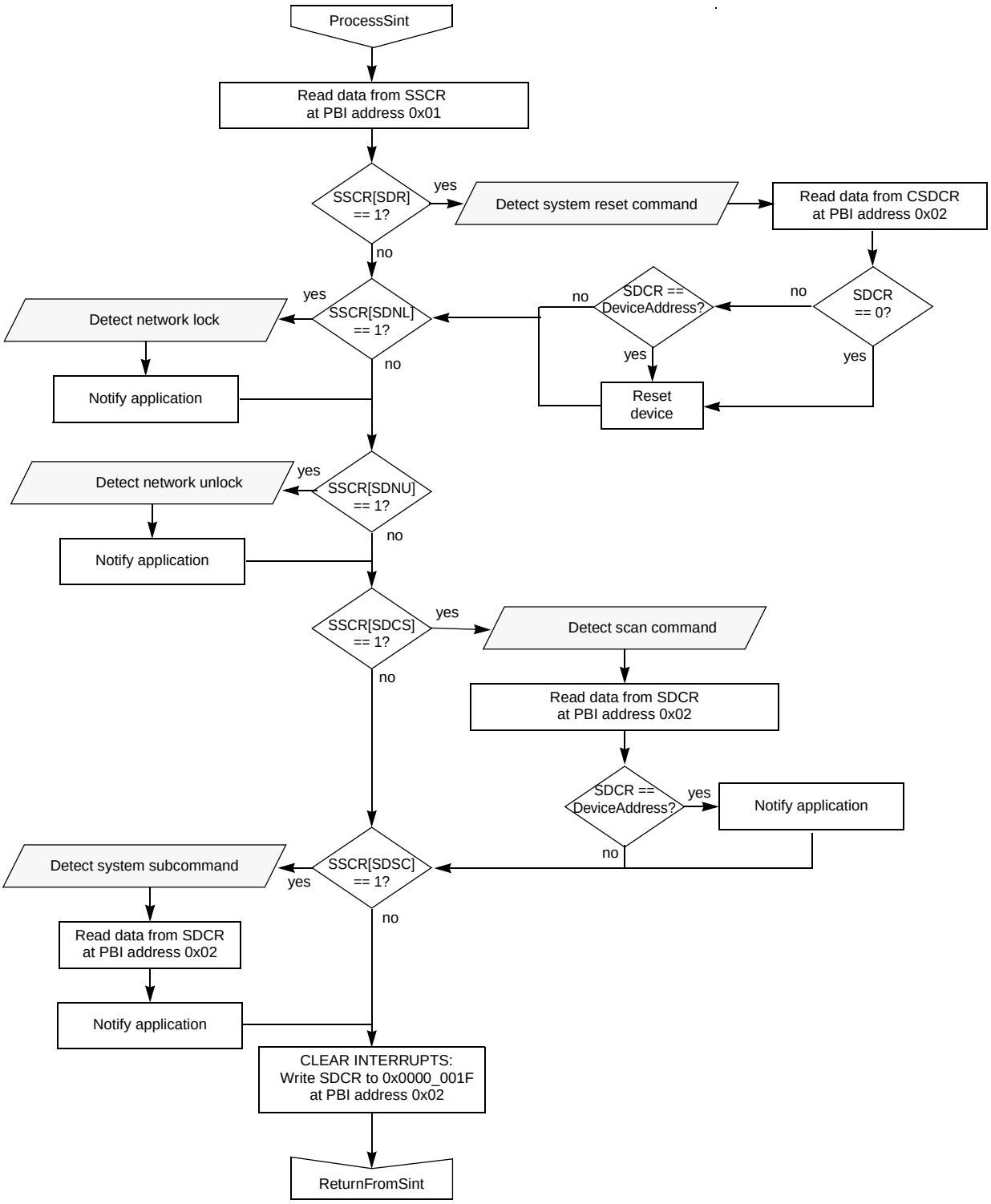


Figure 27-31. System Interrupts

Chapter 28
Enhanced Modular Input/Output Subsystem (eMIOS200)

28.1 Introduction

The eMIOS200 provides functionality to generate or measure time events. The eMIOS200 is implemented with its own configuration of timer channels to suit the target applications needs, while providing a consistent user interface with previous eMIOS implementations. The PXN20 has one eMIOS200 module that implements 16-bit counters.

28.1.1 Block Diagram

Figure 28-1 shows the eMIOS200 block diagram for the PXN20, implementing 24 unified channels. Channels 0, 8, 16, 23, and 24 use channel type A. Channels 1 – 7 and 9 use channel type B. Channels 10 – 15, 17 – 22, and 25 – 31 use channel type C (see Section 28.1.4, eMIOS200 Channel Configurations). The PXN21 implements 32 unified channels in a similar fashion.

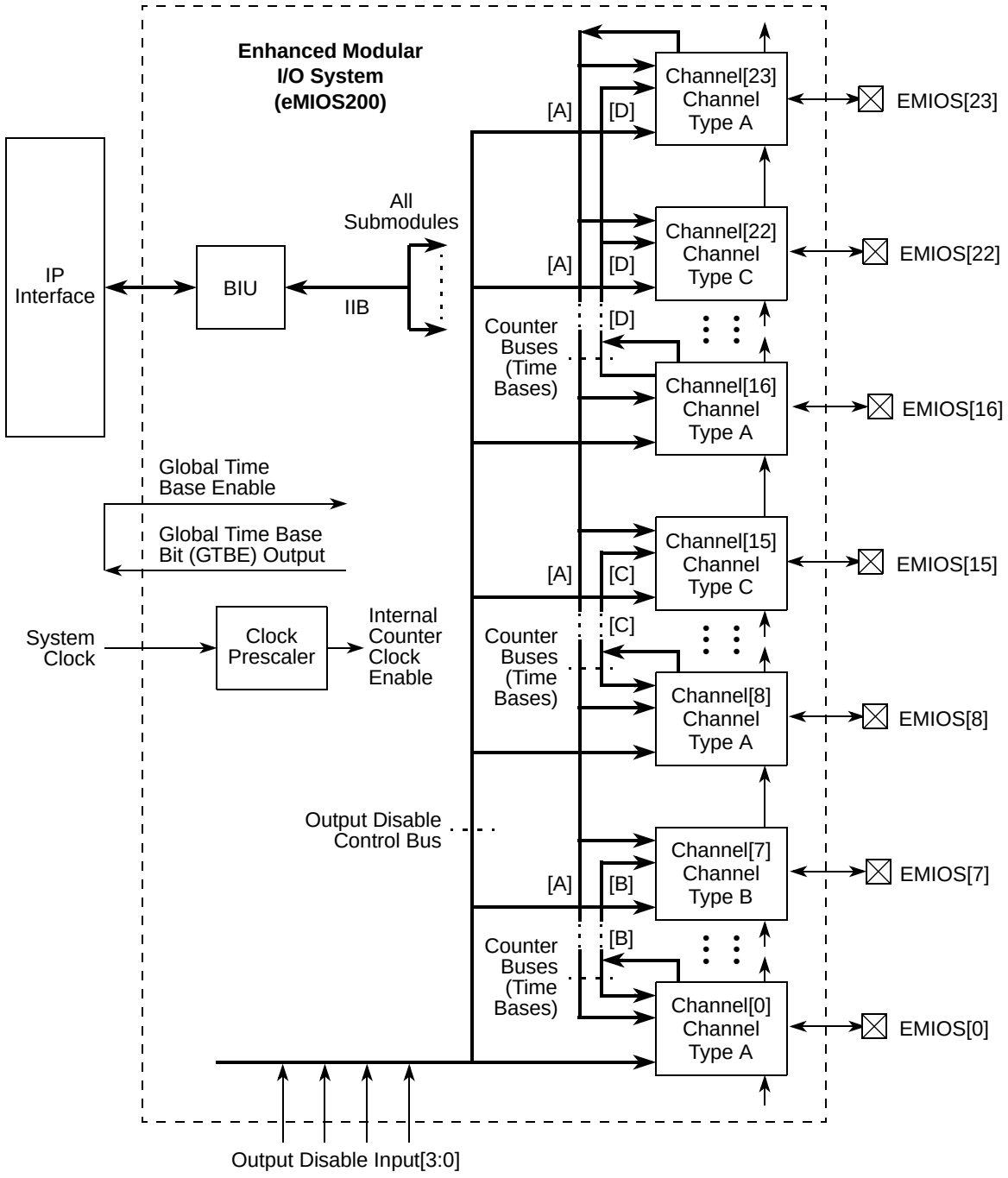


Figure 28-1. eMIOS200 Block Diagram (PXN20)

28.1.2 Features

- As many as 32 channels implemented using three channel types
- Channels features:
  - 16-bit registers for captured/match values

  - 16-bit internal counter
  - Internal prescaler
  - Selectable time base
  - Can generate its own time base
- Four 16-bit-wide counter buses
  - Counter bus A can be driven by unified channel 23
  - Counter buses B, C, D, and E are driven by unified channels 0, 8, 16, and 24, respectively
  - Counter bus A can be shared among all unified channels. UCs 0 to 7, 8 to 15, 16 to 23, and 24 to 31 can share counter buses B, C, D, and E, respectively
- One global prescaler
- The output signal from the module configuration register's global time base enable bit (EMIOS_MCR[GTBE]) is wrapped back into the global timebase enable input so that the timebase of each channel can be started simultaneously.
- Shared time bases through the counter buses
- Shadow FLAG register
- State of eMIOS200 can be frozen for debug purposes
- Debug mode is supported.

28.1.3 Modes of Operation

There are three main operating modes of eMIOS200: run mode, module disable mode, and debug mode. These modes are briefly described in this section.

Run mode is the normal operation mode and is described in Section 28.4, Functional Description.

Module disable mode is used for MCU power management. The clock to the non-memory-mapped logic in the eMIOS200 is stopped while in module disable mode. Module disable mode is entered when MDIS = 1 in the EMIOS_MCR. Individual disabling of the channels is not supported. The eMIOS200 module can also be halted by setting the SIU_HLT0[HLT6] bit (see Section 8.3.2.24, Halt Acknowledge Register (SIU_HLTACKn)).

Debug mode is individually programmed for each channel. When entering this mode, the unified channel registers' contents are frozen, but remain available for read and write access through the IP interface.

28.1.4 eMIOS200 Channel Configurations

Three different types of eMIOS200 channels are implemented on the PXN20. All channels implement the General Purpose Input Output mode (GPIO) and the Single Input Capture and Output Compare modes (SAIC, SAOC) in addition to the modes listed below.

Figure 28-2 shows the eMIOS200 channel implementations for the device. Channels 0, 8, 16, 23, and 24 use channel type A. Channels 1 – 7 and 9 use channel type B. Channels 10 – 15, 17 – 22, and 25 – 31 use channel type C.

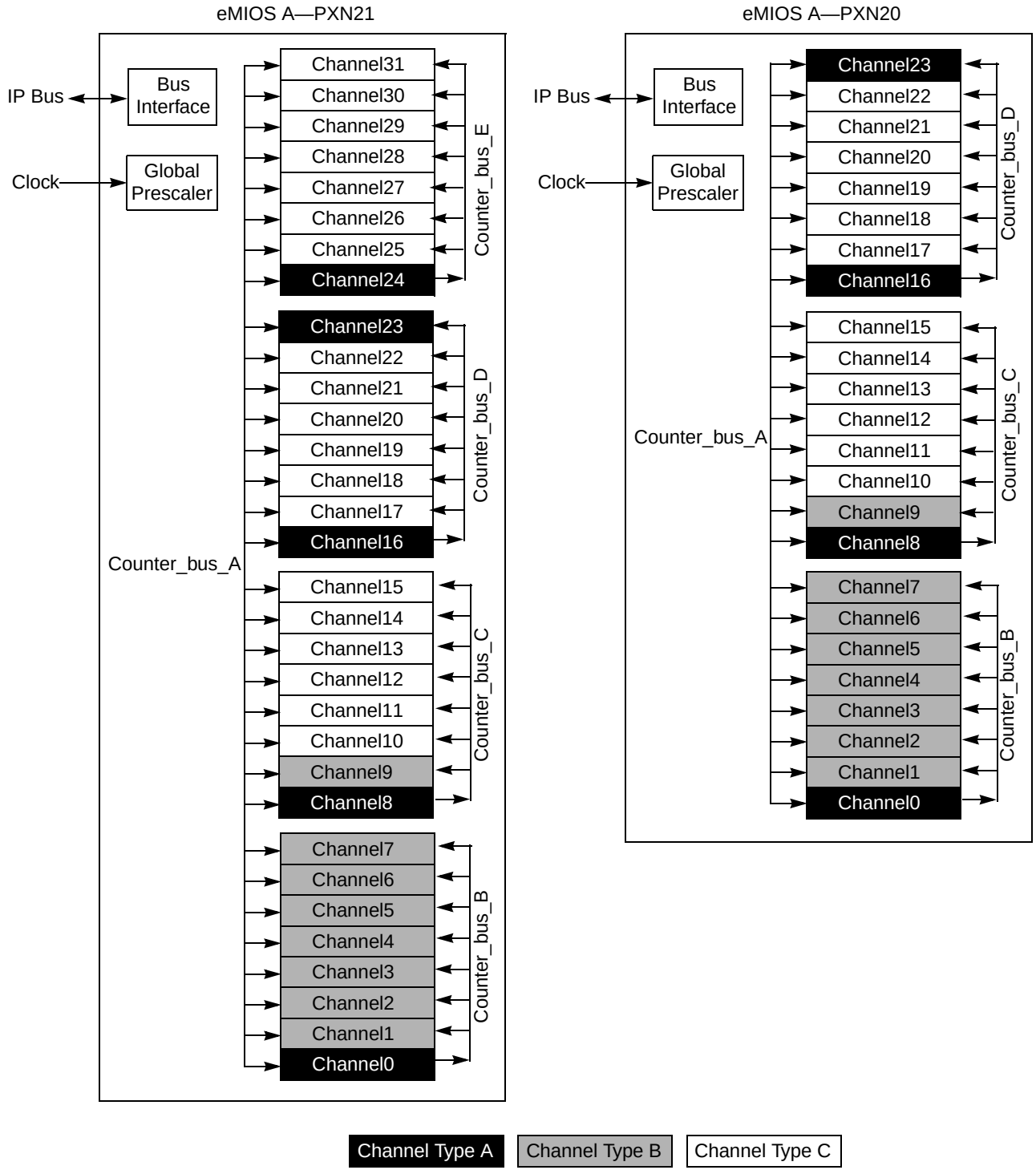


Figure 28-2. eMIOS200 Channel Configuration

28.1.4.1 Type A: Counter Channels

This channel type implements the counters used to drive the module counter buses, and is also available to provide additional dedicated functions such as pulse edge counting, the lighting OPWMT mode,

accumulation (PEC, PEA) and Quadrature Decode (QDEC). On the PXN21, five of these channels are implemented on the device. On the PXN20, four of these channels are implemented.

28.1.4.2 Type B: Complex Channels

These complex channel types offer most of the modes already available on the counter channels. This channel type includes Center aligned PWM modes with deadtime to allow support for motor control applications, and may be combined with the quadrature decode of Type A channels. This channel type also supports the lighting OPWMT mode, to add to the number of channels able to support this function as high end BCM controllers need many of these channel types. Eight of these channels are implemented on both versions of the PXN20 device.

28.1.4.3 Type C: Lighting Channels

The majority of the timer channels are implemented using this type of channel. Its prime role is support for lighting control with the provision of the OPWMT mode, but also allows some other simple timed I/O functionality to be provided. On the PXN21, 19 of these channels are implemented on the device. On the PXN20, 12 of these channels are implemented.

Table 28-1. Supported Modes on PXN20 eMIOS Modules

Description	Name	Channel Type			Number Supported		Section/Page
		Type A	Type B	Type C	PXN21	PXN20	
General Purpose Input / Output	GPIO	X	X	X	32	24	28.4.1.1.1/28-22
Single Action Input Capture	SAIC	X	X	X	32	24	28.4.1.1.2/28-23
Single Action Output Compare	SAOC	X	X	X	32	24	28.4.1.1.3/28-24
Input Pulse-Width Measurement	IPWM	X	X	X	32	24	28.4.1.1.4/28-25
Input Period Measurement	IPM	X	X	X	32	24	28.4.1.1.5/28-27
Double Action Output Compare	DAOC	X	X	X	32	24	28.4.1.1.6/28-29
Pulse Edge Accumulation	PEA	X	—	—	5	4	28.4.1.1.7/28-31
Pulse Edge Counting	PEC	X	—	—	5	4	28.4.1.1.8/28-32
Quadrature Decode	QDEC	X	—	—	5	4	28.4.1.1.9/28-34
Modulus Counter	MC	X	X	—	13	12	28.4.1.1.10/28-35
Modulus Counter Buffered (Up / Down)	MCB	X	X	—	13	12	28.4.1.1.11/28-37
Output Pulse Width and Frequency Modulation Buffered	OPWFMB	X	X	—	13	12	28.4.1.1.12/28-40
Center-Aligned Output PWM Buffered with Dead Time	OPWMCB	X	X	—	13	12	28.4.1.1.13/28-45
Output Pulse Width Modulation Buffered	OPWMB	X	X	X	32	24	28.4.1.1.14/28-50
Output Pulse Width Modulation Trigger	OPWMT	X	X	X	32	24	28.4.1.1.15/28-53
Input Filter	IPF	X	X	X	32	24	28.4.1.2/28-57

28.2 External Signal Description

Refer to Table 3-1 and Section 3.4, Detailed Signal Description, for detailed signal descriptions.

Each channel has one external signal, eMIOS[*n*]. Through the pad configuration register (SIU_PCR*n*[PA]), you can choose to have a pin's function be the eMIOS channel in either or both places as described in Table 28-6.

The output disable input [3:0] is provided to implement the output disable feature. They are connected to emios_flag_out signals according to Section 28.2.2, Output Disable Input — eMIOS200 Output Disable Input Signal.

28.2.1 eMIOS[*n*]

eMIOS[*n*] are the eMIOS channel pins. When used as input, an eMIOS[*n*] signal is available to be read by the MCU through the EMIOS_CSR*n*[UCIN]. When used as output, eMIOS[*n*] signal is configured in the unified channel status and control register (EMIOS_CSR*n*).

NOTE

All eMIOS channels support both input and output functions. When the eMIOS function is the primary function of a pin, then both the input and output functions are supported. When the eMIOS function is not the primary function of the pin, then only the output functions are supported.

28.2.2 Output Disable Input — eMIOS200 Output Disable Input Signal

Output disable inputs are connected as defined in Table 28-2.

Table 28-2. ODIS Input Signals

eMIOS200 channel	Output Disable Input Signal
emios_flag_out[20]	Output disable input[3]
emios_flag_out[19]	Output disable input[2]
emios_flag_out[18]	Output disable input[1]
emios_flag_out[17]	Output disable input[0]

28.3 Memory Map and Register Description

This section provides a detailed description of all eMIOS200 registers.

28.3.1 Memory Map

The eMIOS200 memory map is shown in Table 28-3. The address of each register is given as an offset to the eMIOS200 base address. Registers are listed in address order, identified by complete name and mnemonic, and lists the type of accesses allowed.

Table 28-3. eMIOS200 Memory Map

Offset from EMIOS_BASE (0xFFFE_4000)	Register	Access[1]	Reset Value	Section/Page
Global Registers				
0x0000	EMIOS_MCR—Module Configuration Register	R/W	0x0000_0000	28.3.2.1/28-9
0x0004	EMIOS_GFR—Global FLAG Register	R	0x0000_0000	28.3.2.2/28-10
0x0008	EMIOS_OUDR—Output Update Disable Register	R/W	0x0000_0000	28.3.2.3/28-11
0x000C	EMIOS_UCDIS—Stop (Disable) Channel Register	R/W	0x0000_0000	28.3.2.4/28-11
0x0010–0x001F	Reserved			
Unified Channel 0 Registers				
0x0020	EMIOS_CADR[0]—Channel A Data Register	R/W	0x0000_0000	28.3.2.5/28-12
0x0024	EMIOS_CBDR[0]—Channel B Data Register	R/W	0x0000_0000	28.3.2.6/28-12
0x0028	EMIOS_CCNTR[0]—Channel Counter Register	R	0x0000_0000	28.3.2.7/28-13
0x002C	EMIOS_CCR[0]—Channel Control Register	R/W	0x0000_0000	28.3.2.8/28-14
0x0030	EMIOS_CSR[0]—Channel Status Register	R	0x0000_0000	28.3.2.9/28-19
0x0034	EMIOS_ALTA[0][2]—Alternate A Register	R/W	0x0000_0000	28.3.2.10/28-20
0x0038–0x003F	Reserved			
Unified Channel 1 Registers				
0x0040	EMIOS_CADR[1]—A Register	R/W	0x0000_0000	28.3.2.5/28-12
0x0044	EMIOS_CBDR[1]—B Register	R/W	0x0000_0000	28.3.2.6/28-12
0x0048	EMIOS_CCNTR[1]—Counter Register	R	0x0000_0000	28.3.2.7/28-13
0x004C	EMIOS_CCR[1]—Control Register	R/W	0x0000_0000	28.3.2.8/28-14
0x0050	EMIOS_CSR[1]—Status Register	R	0x0000_0000	28.3.2.9/28-19
0x0054	EMIOS_ALTA[1][2]—Alternate A Register	R/W	0x0000_0000	28.3.2.10/28-20
0x0058–0x005F	Reserved			
Unified Channel 2 Registers				
0x0060	EMIOS_CADR[2]—A Register	R/W	0x0000_0000	28.3.2.5/28-12
0x0064	EMIOS_CBDR[2]—B Register	R/W	0x0000_0000	28.3.2.6/28-12
0x0068	EMIOS_CCNTR[2]—Counter Register	R	0x0000_0000	28.3.2.7/28-13
0x006C	EMIOS_CCR[2]—Control Register	R/W	0x0000_0000	28.3.2.8/28-14
0x0070	EMIOS_CSR[2]—Status Register	R	0x0000_0000	28.3.2.9/28-19
0x0074	EMIOS_ALTA[2][2]—Alternate A Register	R/W	0x0000_0000	28.3.2.10/28-20
0x0078–0x007F	Reserved			

Table 28-3. eMIOS200 Memory Map (continued)

Offset from EMIOS_BASE (0xFFFE_4000)	Register	Access[1]	Reset Value	Section/Page
Unified Channel 3–31 Registers[3]				
0x0080–0x041F	Same as other Channel Registers (e.g. EMIOS_CADR[2], EMIOS_CBDR[2], etc.)	—	—	—
0x0420–0x3FFF	Reserved			

1 Note that R/W registers may contain some read-only or write-only bits.

2 The alternate address register provides and alternate read-only address to access A2 channel registers in pulse edge counting (PEC) and windowed programmable time accumulation (WPTA) modes. IF EMIOS_CADR[*n*] register is used with EMIOS_ALTA[*n*], both A1 and A2 registers can be accessed in these modes.

3 For a complete list of Unified Channel registers with their addresses, please see eMIOS_A in Appendix A, Memory Map.

Table 28-4. Unified Channel Base Offsets

Unified Channel	Offset from EMIOS_BASE (0xFFFE_4000)	Unified Channel	Offset from EMIOS_BASE (0xFFFE_4000)
Unified Channel 0	0x0020	Unified Channel 16	0x0220
Unified Channel 1	0x0040	Unified Channel 17	0x0240
Unified Channel 2	0x0060	Unified Channel 18	0x0260
Unified Channel 3	0x0080	Unified Channel 19	0x0280
Unified Channel 4	0x00A0	Unified Channel 20	0x02A0
Unified Channel 5	0x00C0	Unified Channel 21	0x02C0
Unified Channel 6	0x00E0	Unified Channel 22	0x02E0
Unified Channel 7	0x0100	Unified Channel 23	0x0300
Unified Channel 8	0x0120	Unified Channel 24	0x0320
Unified Channel 9	0x0140	Unified Channel 25	0x0340
Unified Channel 10	0x0160	Unified Channel 26	0x0360
Unified Channel 11	0x0180	Unified Channel 27	0x0380
Unified Channel 12	0x01A0	Unified Channel 28	0x03A0
Unified Channel 13	0x01C0	Unified Channel 29	0x03C0
Unified Channel 14	0x01E0	Unified Channel 30	0x03E0
Unified Channel 15	0x0200	Unified Channel 31	0x0400

28.3.2 Register Descriptions

This section lists the eMIOS200 registers in address order and describes the registers and their bit fields.

28.3.2.1 eMIOS200 Module Configuration Register (EMIOS_MCR)

The EMIOS_MCR contains global control bits for the eMIOS200 block.

Offset: EMIOS_BASE + 0x0000 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R		MDIS	FRZ	GTBE	0	GPREN	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	GPRE[0:7]								0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 28-3. eMIOS200 Module Configuration Register (EMIOS_MCR)

Table 28-5. EMIOS_MCR Field Descriptions

Field	Description
bit 0	Reserved. **Note:** Writing to this bit updates the register value, and reading it returns the last value written, but the bit has no other effect.
MDIS	Module Disable Bit. Puts the eMIOS200 in low-power mode. The MDIS bit is used to stop the clock of the block, except the access to registers EMIOS_MCR, EMIOS_OUDR, and EMIOS_UCDIS. 0 Clock is running. 1 Enter low-power mode.
FRZ	Freeze Bit. Enables the eMIOS200 to freeze the registers of the unified channels when debug mode is requested at MCU level. Each unified channel must have FREN bit set in order to enter freeze mode. While in freeze mode, the eMIOS200 continues to operate to allow the MCU access to the unified channel registers. The unified channel remains frozen until the FRZ bit is written to 0 or the MCU exits debug mode or the unified channel FREN bit is cleared. 0 Exit freeze mode. 1 Stops unified channel operation when in debug mode and the FREN bit is set in the EMIOS_CCR[*n*] register.
GTBE	Global Time Base Enable Bit. The GTBE bit is used to export a global time base enable from the module and provide a method to start time bases of several blocks simultaneously. 0 Global time base enable out signal negated. 1 Global time base enable out signal asserted. **Note:** The global time base enable input pin controls the internal counters. When asserted, internal counters are enabled. When negated, internal counters are disabled.

Table 28-5. EMIOS_MCR Field Descriptions (continued)

Field	Description
GPREN	Global Prescaler Enable Bit. The GPREN bit enables the prescaler counter. 0 Prescaler disabled (no clock) and prescaler counter is cleared. 1 Prescaler enabled.
GPRE	Global Prescaler Bits. The GPRE bits select the clock divider value for the global prescaler.

GPRE	Divide Ratio
0000_0000	1
0000_0001	2
0000_0010	3
0000_0011	4
. . .	. . .
1111_1110	255
1111_1111	256

28.3.2.2 eMIOS200 Global Flag Register (EMIOS_GFR)

Offset: EMIOS_BASE + 0x0004 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	F31	F30	F29	F28	F27	F26	F25	F24	F23	F22	F21	F20	F19	F18	F17	F16
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	F15	F14	F13	F12	F11	F10	F9	F8	F7	F6	F5	F4	F3	F2	F1	F0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 28-4. eMIOS200 Global Flag Register (EMIOS_GFR)

Table 28-6. EMIOS_GFR Field Descriptions

Field	Description
F[31:0]	FLAG Bits 23–0. The EMIOS_GFR is a read-only register that groups the FLAG bits from all channels. This organization improves interrupt handling on simpler devices. These bits are mirrors of the FLAG bits of each channel register (EMIOS_CSR*n*). The PXN21 implements all 32 channels, comprising bits F31 through F0. The PXN20 implements only 24 channels, comprising bits F23 through F0.

28.3.2.3 eMIOS200 Output Update Disable Register (EMIOS_OUDR)

Offset: EMIOS_BASE + 0x0008 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	OU31	OU30	OU29	OU28	OU27	OU26	OU25	OU24	OU23	OU22	OU21	OU20	OU19	OU18	OU17	OU16
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	OU15	OU14	OU13	OU12	OU11	OU10	OU9	OU8	OU7	OU6	OU5	OU4	OU3	OU2	OU1	OU0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 28-5. eMIOS200 Output Update Disable Register (EMIOS_OUDR)

Table 28-7. EMIOS_OUDR Field Descriptions

Field	Description
OU[31:0]	Channel [*n*] Output Update Disable Bits. When running MCB mode or an output mode, values are written to registers A2 and B2. OU[*n*] bits are used to disable transfers from registers A2 to A1 and B2 to B1. Each bit controls one channel. 0 Transfer enabled. Depending on the operation mode, transfer may occur immediately or in the next period. Unless stated otherwise, transfer occurs immediately. 1 Transfers disabled. **Note:** The PXN21 implements all 32 channels, comprising bits OU31 through OU0. The PXN20 implements only 24 channels, comprising bits OU23 through OU0.

28.3.2.4 eMIOS200 Disable Channel Register (EMIOS_UCDIS)

Offset: EMIOS_BASE + 0x000C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	UCDIS[31:16]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	UCDIS[15:0]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 28-6. eMIOS200 Enable Channel Register (EMIOS_UCDIS)

Table 28-8. EMIOS_UCDIS Field Descriptions

Field	Description
UCDIS[31:0]	Enable Channel [*n*] Bit. The UCDIS[*n*] bit is used to disable each of the unified channels by stopping its respective clock. 0 UC [*n*] enabled. 1 UC [*n*] disabled. **Note:** The PXN21 implements all 32 channels, comprising bits UCDIS31 through UCDIS0. The PXN20 implements only 24 channels, comprising bits UCDIS23 through UCDIS0.

28.3.2.5 eMIOS200 A Register (EMIOS_CADR[*n*])

Offset: UC[*n*] base address + 0x0000 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	A															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 28-7. eMIOS200 A Register (EMIOS_CADR[*n*])

Depending on the mode of operation, internal registers A1 or A2, used for matches and captures, can be assigned to address EMIOS_CADR[*n*]. A1 and A2 are cleared by reset. Table 28-9 summarizes the EMIOS_CADR[*n*] writing and reading accesses for all operation modes. For more information see Section 28.4.1.1, Unified Channel Modes of Operation.

28.3.2.6 eMIOS200 B Register (EMIOS_CBDR[*n*])

Offset: UC[*n*] base address + 0x0004 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	B															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 28-8. eMIOS200 B Register (EMIOS_CBDR[*n*])

Depending on the mode of operation, internal registers B1 or B2 can be assigned to address EMIOS_CBDR[*n*]. Both B1 and B2 are cleared by reset. Table 28-9 summarizes the EMIOS_CBDR writing and reading accesses for all operation modes. For more information see section Section 28.4.1.1, Unified Channel Modes of Operation.

Depending on the channel configuration, it may have EMIOS_CBDR register or not. This means that if at least one mode that requires the register is implemented, then the register is present. Otherwise, it is absent.

Table 28-9. EMIOS_CADR[*n*] and EMIOS_CBDR[*n*] Values Assignment

Operation Mode	Register Access				
	Write	Read	Write	Read	Alternate Read
GPIO	A1, A2	A1	B1, B2	B1	—
SAIC[1]	—	A2	B2	B2	—
SAOC[1]	A2	A1	B2	B2	—
IPWM	—	A2	—	B1	—
IPM	—	A2	—	B1	—
DAOC	A2	A1	B2	B1	—
PEA	A1	A2	—	B1	—
PEC[1]	A1	A1	B1	B1	A2
QDEC[1]	A1	A1	B2	B2	—
MC[1]	A2	A1	B2	B2	—
OPWMT	A1	A1	B2	B1	A2
MCB[1]	A2	A1	B2	B2	—
OPWFMB	A2	A1	B2	B1	—
OPWMCB	A2	A1	B2	B1	—
OPWMB	A2	A1	B2	B1	—

1 In these modes, the register EMIOS_CBDR[*n*] is not used, but B2 can be accessed.

28.3.2.7 eMIOS200 Counter Register (EMIOS_CCNTR[*n*])

Offset: UC[*n*] base address + 0x0008 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	C															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

1 In GPIO mode or freeze action, this register is writable.

Figure 28-9. eMIOS200 Counter Register (EMIOS_CCNTR[*n*])

The EMIOS_CCNTR[*n*] register contains the value of the internal counter for eMIOS channel *n*. When GPIO mode is selected or the channel is frozen, the EMIOS_CCNTR[*n*] register is read/write. For all other modes, the EMIOS_CCNTR[*n*] is a read-only register. When entering some operation modes, this register is automatically cleared (refer to Section 28.4.1.1, Unified Channel Modes of Operation, for details).

Depending on the channel configuration it may have an internal counter or not. It means that if at least one mode that requires the counter is implemented, then the counter is present, otherwise it is absent.

28.3.2.8 eMIOS200 Control Register (EMIOS_CCR[*n*])

Offset: UC[*n*] base address + 0x000C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	FREN	ODIS	ODISSL		UCPRE		UC PREN	DMA	0	IF				FCK	FEN	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	FORC MA	FORC MB	0	BSL		ED SEL	ED POL	MODE[0:6]						
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 28-10. eMIOS200 Control Register (EMIOS_CCR[*n*])

The control register gathers bits reflecting the status of the unified channel input/output signals and the overflow condition of the internal counter, as well as several read/write control bits.

Table 28-10. EMIOS_CCR[*n*] Field Descriptions

Field	Description
FREN	Freeze Enable Bit. The FREN bit, if set and validated by FRZ bit in EMIOS_MCR register, freezes all registers' values when in debug mode, allowing the MCU to perform debug functions. 0 Normal operation. 1 Freeze unified channel registers' values.
ODIS	Output Disable Bit. The ODIS bit allows disabling the output pin when running any of the output modes with the exception of GPIO mode. 0 The output pin operates normally. 1 If the selected output disable input signal is asserted, the output pin goes to EDPOL for OPWFMB and OPWMB modes and to the complement of EDPOL for other modes, but the unified channel continues to operate normally, i.e., it continues to produce FLAG and matches. When the selected output disable input signal is negated, the output pin operates normally.
ODISSL	Output Disable Select Bits. The ODISSL bits select one of the four output disable input signals.

ODISSL	Input Signal
00	Output disable input 0
01	Output disable input 1
10	Output disable input 2
11	Output disable input 3

Table 28-10. EMIOS_CCR[*n*] Field Descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>UCPRE</td><td>Prescaler Bits. The UCPRE bits select the clock divider value for the internal prescaler of unified channel.
<table>
<tr><th>UCPRE</th><th>Divide Ratio</th></tr>
<tr><td>00</td><td>1</td></tr>
<tr><td>01</td><td>2</td></tr>
<tr><td>10</td><td>3</td></tr>
<tr><td>11</td><td>4</td></tr>
</table></td></tr>
<tr><td>UCPREN</td><td>Prescaler Enable Bit. The UCPREN bit enables the prescaler counter.
0 Prescaler disabled (no clock) and prescaler counter is loaded with UCPRE value.
1 Prescaler enabled.</td></tr>
<tr><td>DMA</td><td>Direct Memory Access Bit. The DMA bit selects whether the FLAG generation is used as an interrupt or as a DMA request.
0 FLAG assigned to interrupt request.
1 FLAG assigned to DMA request.</td></tr>
<tr><td>IF</td><td>Input Filter Bits. The IF bits control the programmable input filter, selecting the minimum input pulse width that can pass through the filter. For output modes, these bits have no meaning.
<table>
<tr><th>IF[1]</th><th>Minimum Input Pulse Width [FLT_CLK Periods]</th></tr>
<tr><td>0000</td><td>Bypassed[2]</td></tr>
<tr><td>0001</td><td>02</td></tr>
<tr><td>0010</td><td>04</td></tr>
<tr><td>0100</td><td>08</td></tr>
<tr><td>1000</td><td>16</td></tr>
<tr><td>All others</td><td>Reserved</td></tr>
</table>
1 Filter latency is three clock edges.
2 The input signal is synchronized before arriving to the digital filter.</td></tr>
<tr><td>FCK</td><td>Filter Clock Select Bit. The FCK bit selects the clock source for the programmable input filter.
0 Prescaled clock.
1 Main clock.</td></tr>
<tr><td>FEN</td><td>FLAG Enable Bit. The FEN bit allows the unified channel FLAG bit to generate an interrupt signal or a DMA request signal (the type of signal to be generated is defined by the DMA bit).
0 Disable (FLAG does not generate an interrupt or DMA request).
1 Enable (FLAG generates an interrupt or DMA request).</td></tr>
<tr><td>FORCMA</td><td>Force Match A Bit. For output modes, the FORCMA bit is equivalent to a successful comparison on comparator A (except that the FLAG bit is not set). This bit is cleared by reset and is always read as 0. This bit is valid for every output operation mode which uses comparator A, otherwise it has no effect.
0 Has no effect.
1 Force a match at comparator A.
For input modes, the FORCMA bit is not used and writing to it has no effect.</td></tr>
</table>

Table 28-10. EMIOS_CCR[*n*] Field Descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>FORCMB</td><td>Force Match B Bit. For output modes, the FORCMB bit is equivalent to a successful comparison on comparator B (except that the FLAG bit is not set). This bit is cleared by reset and is always read as 0. This bit is valid for every output operation mode which uses comparator B, otherwise it has no effect.
0 Has no effect.
1 Force a match at comparator B.
For input modes, the FORCMB bit is not used and writing to it has no effect.</td></tr>
<tr><td>BSL</td><td>Bus Select Bits. The BSL bits are used to select either one of the counter buses or the internal counter to be used by the unified channel.
<table>
<tr><th>BSL</th><th>Selected Bus</th></tr>
<tr><td>00</td><td>All channels: counter bus[A]</td></tr>
<tr><td>01</td><td>Channels 0 to 7: counter bus[B]
Channels 8 to 15: counter bus[C]
Channels 16 to 23: counter bus[D]
Channels 24 to 31: counter bus[E]</td></tr>
<tr><td>10</td><td>Reserved</td></tr>
<tr><td>11</td><td>All channels: internal counter</td></tr>
</table></td></tr>
<tr><td>EDSEL</td><td>Edge Selection Bit. For input modes, the EDSEL bit selects if the internal counter is triggered by both edges of a pulse or by a single edge only as defined by the EDPOL bit. When not shown in the mode of operation description, this bit has no effect.
0 Single edge triggering defined by the EDPOL bit.
1 Both edges triggering.

For GPIO in mode, the EDSEL bit selects if a FLAG can be generated.
0 A FLAG is generated as defined by the EDPOL bit.
1 No FLAG is generated.

For SAOC mode, the EDSEL bit selects the behavior of the output flip-flop at each match.
0 The EDPOL value is transferred to the output flip-flop.
1 The output flip-flop is toggled.</td></tr>
<tr><td>EDPOL</td><td>Edge Polarity Bit. For input modes, the EDPOL bit asserts which edge triggers either the internal counter or an input capture or a FLAG. When not shown in the mode of operation description, this bit has no effect.
0 Trigger on a falling edge.
1 Trigger on a rising edge.

For output modes, the EDPOL bit is used to select the logic level on the output pin.
0 A match on comparator A clears the output flip-flop, while a match on comparator B sets it.
1 A match on comparator A sets the output flip-flop, while a match on comparator B clears it.</td></tr>
<tr><td>MODE[0:6]</td><td>Mode Selection Bits. The MODE bits select the mode of operation of the unified channel, as shown in Table 28-11. Refer to Table 28-1 for more information on the different modes.

Note: If a reserved value is written to MODE, the results are unpredictable.</td></tr>
</table>

Table 28-11. MODE Bits

MODE[0:6]	Mode	Description
000_0000	GPIO (input)	General Purpose Input/Output mode (input)
000_0001	GPIO (output)	General Purpose Input/Output mode (output)
000_0010	SAIC	Single Action Input Capture
000_0011	SAOC	Single Action Output Compare
000_0100	IPWM	Input Pulse Width Measurement
000_0101	IPM	Input Period Measurement
000_0110	DAOC	Double Action Output compare (with FLAG set on B match)
000_0111	DAOC	Double Action Output compare (with FLAG set on both match)
000_1000	PEA	Pulse/Edge Accumulation (continuous)
000_1001	PEA	Pulse/Edge Accumulation (single shot)
000_1010	PEC	Pulse/Edge Counting (continuous)
000_1011	PEC	Pulse/Edge Counting (single shot)
000_1100	QDEC	Quadrature Decode (for count & direction encoders type)
000_1101	QDEC	Quadrature Decode (for phase_A & phase_B encoders type)
000_1110 – 000_1111	Reserved	
001_0000	MC	Modulus Counter (Up counter with clear on match start, internal clock)
001_0001	MC	Modulus Counter (Up counter with clear on match start, external clock)
001_0010	MC	Modulus Counter (Up counter with clear on match end, internal clock)
001_0011	MC	Modulus Counter (Up counter with clear on match end, external clock)
001_0100	MCB	Modulus Counter (Up/Down counter with flag on A1 match, internal clock)
001_0101	MCB	Modulus Counter (Up/Down counter with flag on A1 match, external clock)
001_0110	MCB	Modulus Counter (Up/Down counter with flag on A1 match or cycle boundary, internal clock)
001_0111	MCB	Modulus Counter (Up/Down counter with flag on A1 match or cycle boundary, external clock)
001_1000	OPWFMB	Output Pulse Width and Frequency Modulation (flag on B1 match, immediate update)
001_1001	Reserved	
001_1010	OPWFMB	Output Pulse Width and Frequency Modulation (flag on A1 or B1 matches, immediate update)
001_1011	Reserved	
001_1100	OPWMCB	Center Aligned Output Pulse Width Modulation (flag in trailing edge, trail edge dead-time)
001_1101	OPWMCB	Center Aligned Output Pulse Width Modulation (flag in trailing edge, lead edge dead-time)

Table 28-11. MODE Bits (continued)

MODE[0:6]	Mode	Description
001_1110	OPWMCB	Center Aligned Output Pulse Width Modulation (flag in both edges, trail edge dead-time)
001_1111	OPWMCB	Center Aligned Output Pulse Width Modulation (flag in both edges, lead edge dead-time)
010_0000	OPWMB	Output Pulse Width Modulation (flag on B1 match, immediate update)
010_0001	OPWMB	Output Pulse Width Modulation (next period update)
010_0010	OPWMB	Output Pulse Width Modulation (immediate update)
010_0011	OPWMB	Output Pulse Width Modulation (next period update)
010_0100 – 010_0101	Reserved	
010_0110	OPWMT	Output Pulse Width Modulation with Trigger
010_0111 – 100_1111	Reserved	
101_0000	MC	Modulus Counter (Up counter with clear on match start, internal clock)
101_0001	MC	Modulus Counter (Up counter with clear on match start, external clock)
101_0010 – 101_0011	Reserved	
101_0100	MCB	Modulus Counter (Up/Down counter with flag on A1 match, internal clock)
101_0101	MCB	Modulus Counter (Up/Down counter with flag on A1 match, external clock)
101_0110	MCB	Modulus Counter (Up/Down counter with flag on A1 match or cycle boundary, internal clock)
101_0111	MCB	Modulus Counter (Up/Down counter with flag on A1 match or cycle boundary, external clock)
101_1000	OPWFMB	Output Pulse Width and Frequency Modulation Buffered, (flag on B1 match)
101_1001	Reserved	
101_1010	OPWFMB	Output Pulse Width and Frequency Modulation Buffered, (flag on A1or B1 matches)
101_1011	Reserved	
101_1100	OPWMCB	Center Aligned Output Pulse Width Modulation Buffered (flag in trailing edge, trail edge dead-time)
101_1101	OPWMCB	Center Aligned Output Pulse Width Modulation Buffered (flag in trailing edge, lead edge dead-time)
101_1110	OPWMCB	Center Aligned Output Pulse Width Modulation Buffered (flag in both edges, trail edge dead-time)
101_1111	OPWMCB	Center Aligned Output Pulse Width Modulation Buffered (flag in both edges, lead edge dead-time)
110_0000	OPWMB	Output Pulse Width Modulation Buffered (flag on B1 match)
110_0001	Reserved	
110_0010	OPWMB	Output Pulse Width Modulation Buffered (flag on A1or B1 matches)
110_0011 – 111_1111	Reserved	

28.3.2.9 eMIOS200 Status Register (EMIOS_CSR[*n*])

Offset: UC[*n*] base address + 0x0010 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	OVR	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	w1c															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	OVFL	0	0	0	0	0	0	0	0	0	0	0	0	UCIN	UCOUT	FLAG
W	w1c															w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 28-11. eMIOS200 Status Register (EMIOS_CSR[*n*])

Table 28-12. EMIOS_CSR[*n*] Field Descriptions

Field	Description
OVR	Overrun Bit. The OVR bit indicates that FLAG generation occurred when the FLAG bit was already set. This bit can be cleared by clearing the FLAG bit or by software writing a 1. 0 Overrun has not occurred. 1 Overrun has occurred.
OVFL	Overflow Bit. The OVFL bit indicates that an overflow has occurred in the internal counter. This bit must be cleared by software writing a 1. 0 An overflow has not occurred. 1 An overflow has occurred.
UCIN	Unified Channel Input Pin Bit. The UCIN bit reflects the input pin state after being filtered and synchronized.
UCOUT	Unified Channel Output. The UCOUT bit reflects the output pin state.
FLAG	FLAG Bit. The FLAG bit is set when an input capture or a match event in the comparators occurred. This bit must be cleared by software writing a 1. 0 FLAG cleared. 1 FLAG set event has occurred. **Note:** emios_flag_out reflects the FLAG bit value. When the DMA bit is set, the FLAG bit can be cleared by the DMA controller.

28.3.2.10 eMIOS200 Alternate A Register (EMIOS_ALTA[*n*])

UC[*n*] base address + 0x0014 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	ALTA							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	ALTA															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 28-12. eMIOS200 Alternate A Register (EMIOS_ALTA[*n*])

The EMIOS_ALTA[*n*] register provides an alternate read-only address to access A2 channel registers in PEC and WPTA modes only. If EMIOS_CADR[*n*] register is used with EMIOS_ALTA[*n*], both A1 and A2 registers can be accessed in these modes.

28.4 Functional Description

The three types of channels of the eMIOS200 can operate in the modes as listed in Table 28-1.

The eMIOS200 provides independently operating unified channels (UC) that can be configured and accessed by a host MCU. On the PXN21, as many as five time bases can be shared by the channels through five counter buses. On the PXN20, as many as four time bases can be shared by the channels through four counter buses. Each unified channel can generate its own time base.

The eMIOS200 block is reset at positive edge of the clock (synchronous reset). All registers are cleared on reset.

28.4.1 Unified Channel (UC)

Figure 28-13 shows the unified channel block diagram. Each unified channel consists of:

- Counter bus selector, which selects the time base to be used by the channel for all timing functions
- A programmable clock prescaler
- Two double buffered data registers, A and B, that allow as many as two input capture and/or output compare events to occur before software intervention is needed
- Two comparators (equal only), A and B, which compare the selected counter bus with the value in the data registers
- Internal counter, which can be used as a local time base or to count input events
- Programmable input filter, which ensures that only valid pin transitions are received by channel
- Programmable input edge detector, which detects the rising, falling or either edges
- An output flip-flop, which holds the logic level to be applied to the output pin
- eMIOS200 status and control register

- An output disable input selector, which selects the output disable input signal to be used as output disable

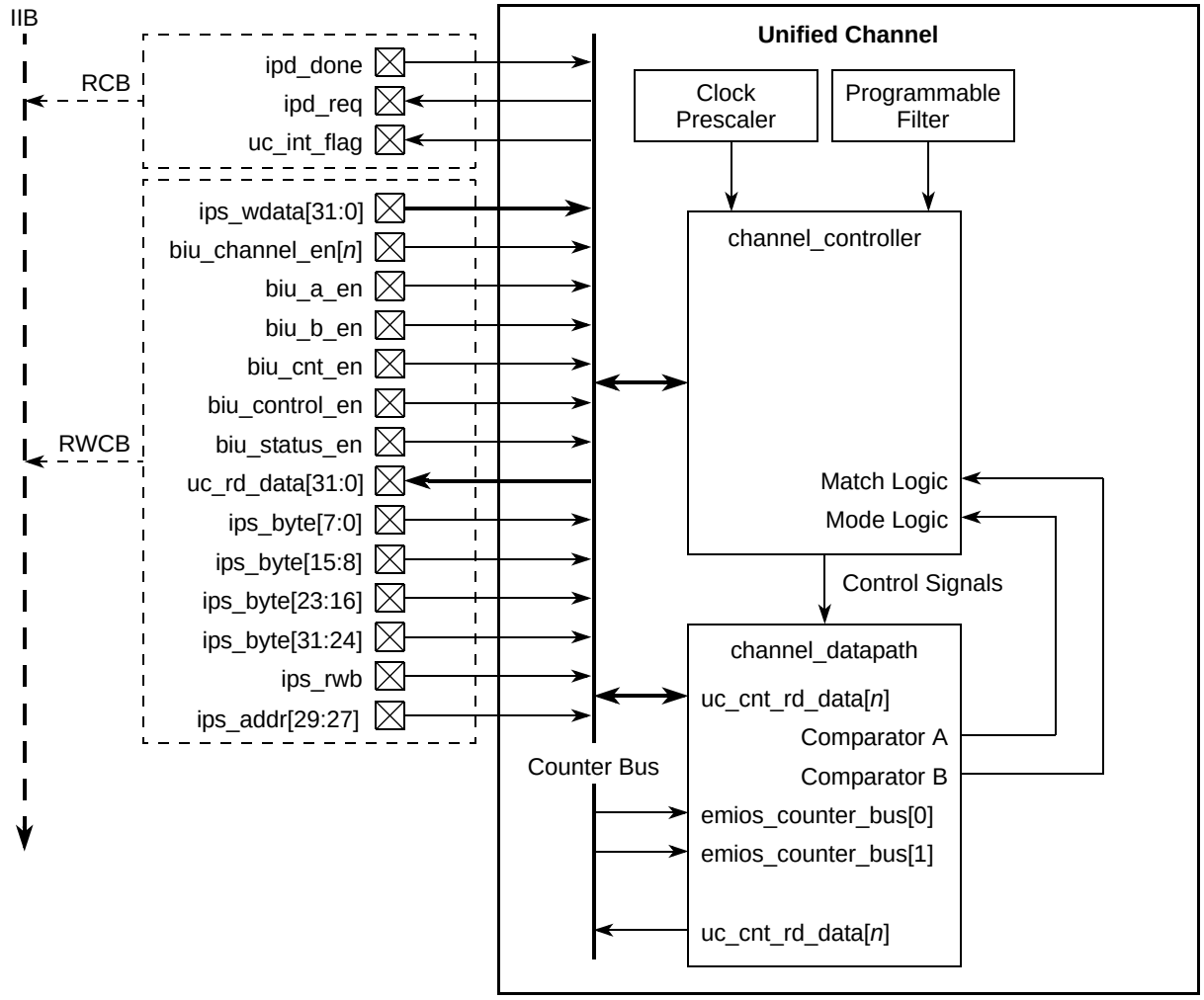


Figure 28-13. Unified Channel Block Diagram

Figure 28-14 shows the unified channel control block diagram.

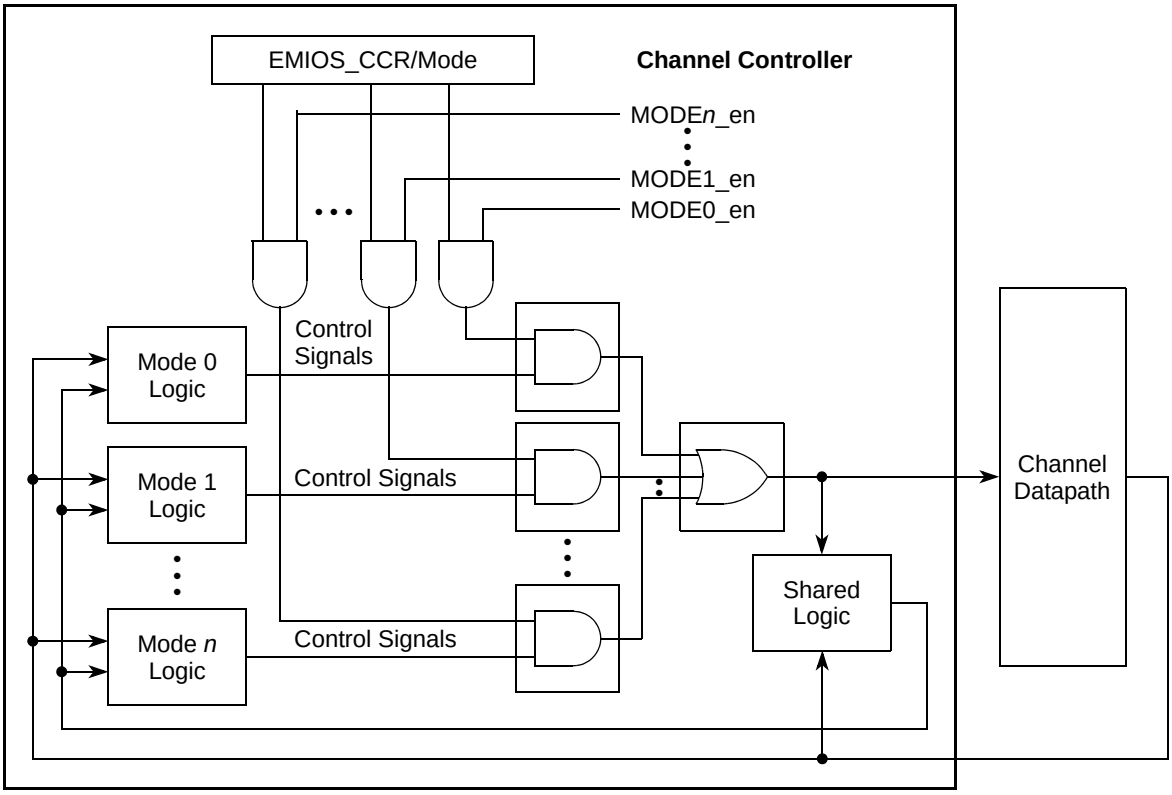


Figure 28-14. Unified Channel Control Block Diagram

28.4.1.1 Unified Channel Modes of Operation

The mode of operation of the unified channel is determined by the mode select bits MODE[0:6] in the EMIOS_CCR[*n*] register (see Table 28-11 for details).

When entering an output mode (except for GPIO mode), the output flip-flop is set to the complement of the EDPOL bit in the EMIOS_CCR[*n*] register.

As the internal counter EMIOS_CCNTR[*n*] continues to run in all modes (except for GPIO mode), it is possible to use this as a time base if the resource is not used in the current mode.

28.4.1.1.1 General-Purpose Input/Output (GPIO) Mode

In GPIO mode, all input capture and output compare functions of the unified channel are disabled, the internal counter (EMIOS_CCNTR[*n*] register) is cleared and disabled. All control bits remain accessible. In order to prepare the unified channel for a new operation mode, writing to registers EMIOS_CADR[*n*] or EMIOS_CBDR[*n*] stores the same value in registers A1/A2 or B1/B2, respectively.

The MODE[6] bit selects between input (MODE[6] = 0) and output (MODE[6] = 1) modes.

When changing the MODE bits, the application software must go to GPIO mode first to reset the unified channel's internal functions properly. Failure to do this could lead to invalid and unexpected output compare or input capture results or the FLAGs being set incorrectly.

In GPIO input mode (MODE = 000_0000),the FLAG generation is determined according to EDPOL and EDSEL bits and the input pin status can be determined by reading the UCIN bit.

In GPIO output mode (MODE = 000_0001), the unified channel is used as a single output port pin and the value of the EDPOL bit is permanently transferred to the output flip-flop.

28.4.1.1.2 Single Action Input Capture (SAIC) Mode

In SAIC mode (MODE = 000_0010), when a triggering event occurs on the input pin, the value on the selected time base is captured into register A2. At the same time, the FLAG bit is set to indicate that an input capture has occurred. Register EMIOS_CADR[*n*] returns the value of register A2. The channel is ready to capture events as soon as SAIC mode is entered coming out from GPIO mode. The events are captured as soon as they occur, thus reading register A always returns the value of the latest captured event. Subsequent captures are enabled with no need of further reads from EMIOS_CADR[*n*] register. The FLAG is set at any time a new event is captured.

The input capture is triggered by a rising, falling or either edges in the input pin, as configured by EDPOL and EDSEL bits in EMIOS_CCR[*n*] register.

Figure 28-15 and Figure 28-16 show how the unified channel can be used for input capture.

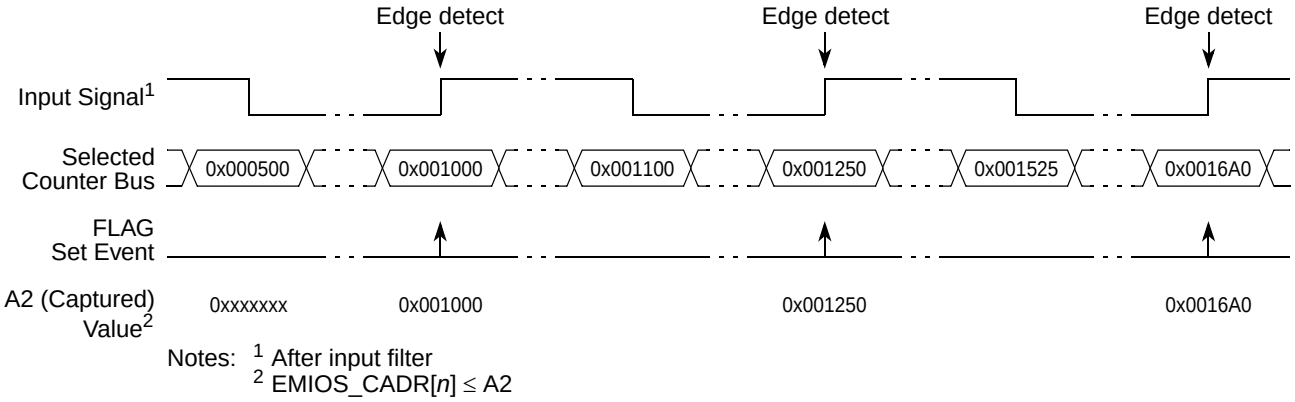


Figure 28-15. Single Action Input Capture with Rising Edge Triggering Example

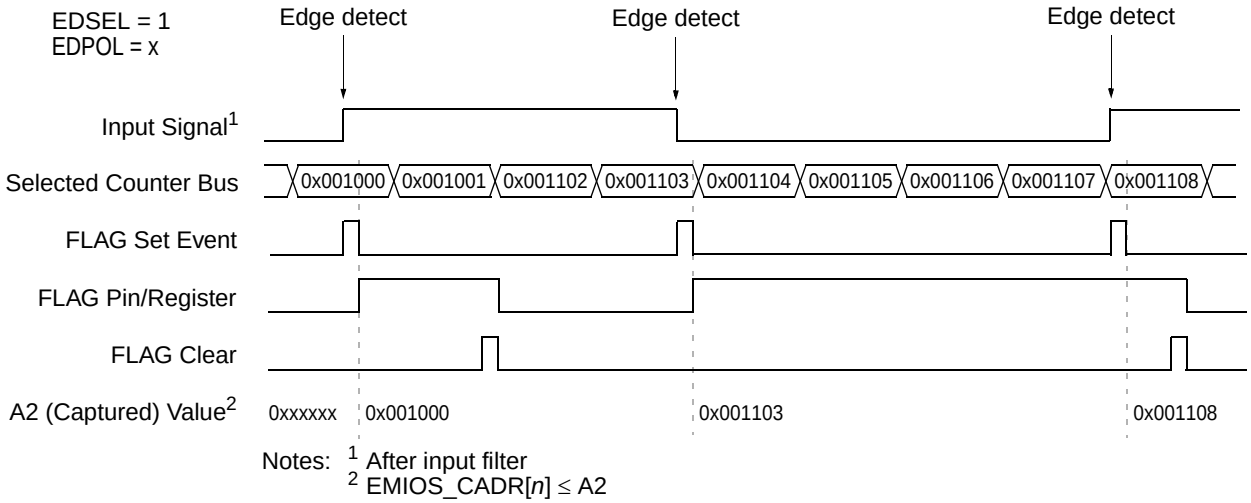


Figure 28-16. Single Action Input Capture with Both Edges Triggering Example

28.4.1.1.3 Single Action Output Compare (SAOC) Mode

In SAOC mode (MODE = 000_0011), a match value is loaded in register A2 and then transferred to register A1 to be compared with the selected time base. When a match occurs, the EDSEL bit selects if the output flip-flop is toggled or if the value in EDPOL is transferred to it. At the same time, the FLAG bit is set to indicate that the output compare match has occurred. Writing to register EMIOS_CADR[*n*] stores the value in register A2 and reading to register EMIOS_CADR[*n*] returns the value of register A1.

An output compare match can be simulated in software by setting the FORCMA bit in EMIOS_CCR[*n*] register. In this case, the FLAG bit is not set.

When SAOC mode is entered coming out from GPIO mode the output flip-flop is set to the complement of the EDPOL bit in the EMIOS_CCR[*n*] register.

Counter bus can be either internal or external and is selected through the BSL bits.

Figure 28-17 and Figure 28-18 show how the unified channel can be used to perform a single output compare with EDPOL value being transferred to the output flip-flop and toggling the output flip-flop at each match, respectively.

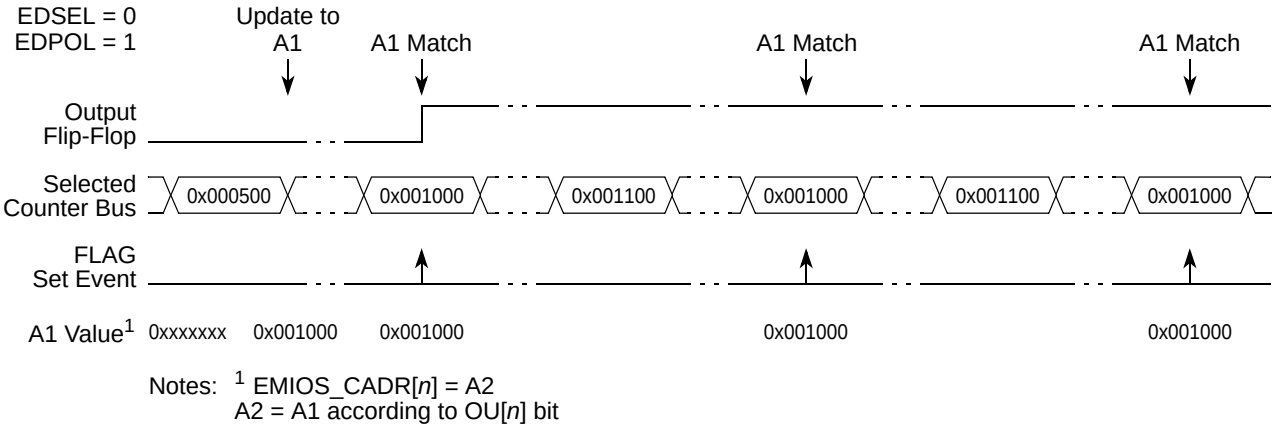


Figure 28-17. SAOC Example—EDPOL Value Being Transferred to the Output Flip-flop

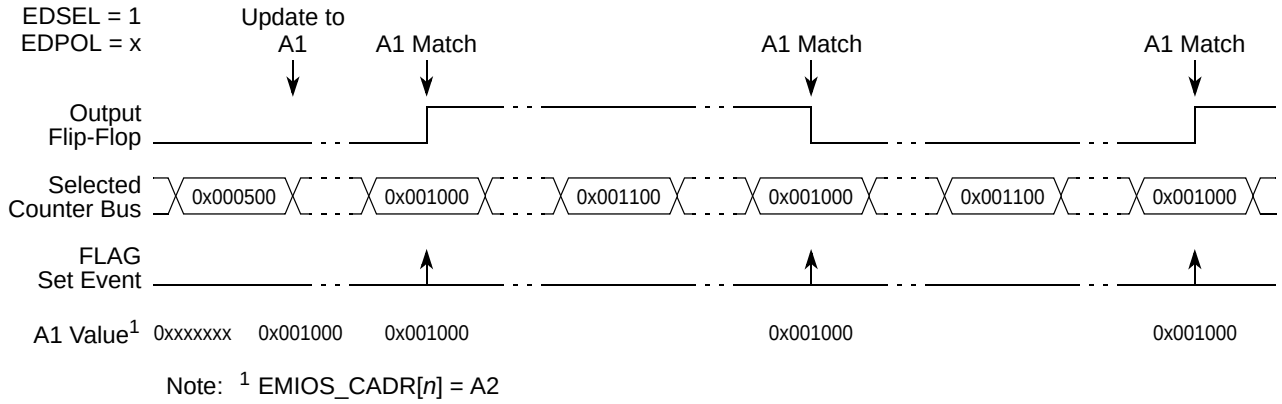


Figure 28-18. SAOC Example —Toggling the Output Flip-Flop

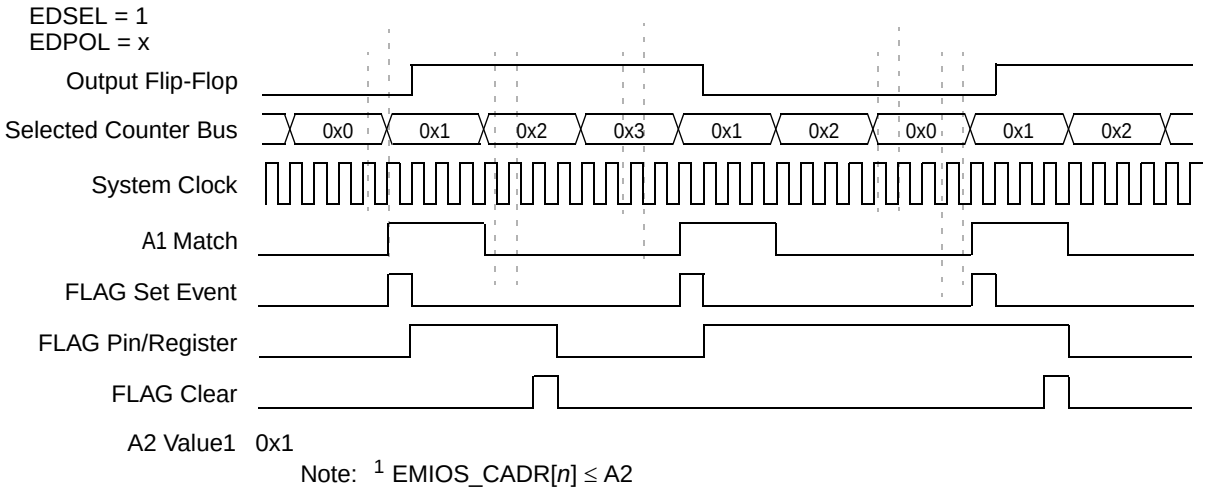


Figure 19. SAOC Example with Flag Behavior

28.4.1.1.4 Input Pulse-Width Measurement (IPWM) Mode

The IPWM mode (MODE = 000_0100) allows the measurement of the width of a positive or negative pulse by capturing the leading edge on register B1 and the trailing edge on register A2. Successive captures are done on consecutive edges of opposite polarity. The leading edge sensitivity (i.e., pulse polarity) is selected by EDPOL bit in the EMIOS_CCR[*n*] register. Registers EMIOS_CADR[*n*] and EMIOS_CBDR[*n*] return the values in register A2 and B1, respectively.

The capture function of register A2 remains disabled until the first leading edge triggers the first input capture on register B2. When this leading edge is detected, the count value of the selected time base is latched into register B2; the FLAG bit is not set. When the trailing edge is detected, the count value of the selected time base is latched into register A2 and, at the same time, the FLAG bit is set and the content of register B2 is transferred to register B1 and to register A1.

If subsequent input capture events occur while the corresponding FLAG bit is set, registers A2, B1, and A1 are updated with the latest captured values and the FLAG remains set. Registers EMIOS_CADR[*n*] and EMIOS_CBDR[*n*] return the value in registers A2 and B1, respectively.

In order to guarantee coherent access, reading EMIOS_CADR[*n*] forces B1 to be updated with the content of register A1. At the same time transfers between B2 and B1 are disabled until the next read of EMIOS_CBDR[*n*] register. Reading EMIOS_CBDR[*n*] register forces B1 be updated with A1 register content and re-enables transfers from B2 to B1, to take effect at the next trailing edge capture. Transfers from B2 to A1 are not blocked at any time.

The input pulse width is calculated by subtracting the value in B1 from A2.

Figure 28-20 shows how the unified channel can be used for input pulse-width measurement.

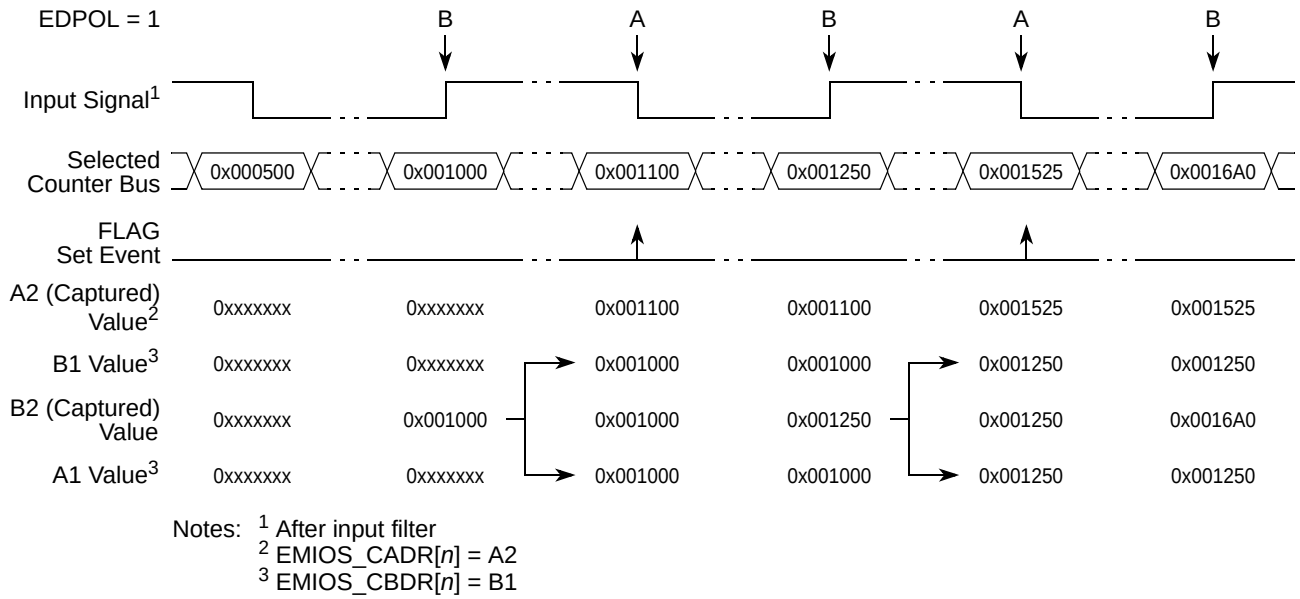


Figure 28-20. Input Pulse-Width Measurement Example

Figure 28-21 shows the A1 and B1 updates when EMIOS_CADR[*n*] and EMIOS_CBDR[*n*] register reads occur. The A1 register has always coherent data related to A2 register. When EMIOS_CADR[*n*] read is performed, the B1 register is loaded with the A1 register content. This guarantees that the data in register B1 always has the coherent data related to the last EMIOS_CADR[*n*] read. The B1 register updates remain locked until EMIOS_CBDR[*n*] read occurs. If EMIOS_CADR[*n*] read is performed, B1 is updated with A1 register content even if the B1 update is locked by a previous EMIOS_CADR[*n*] read operation.

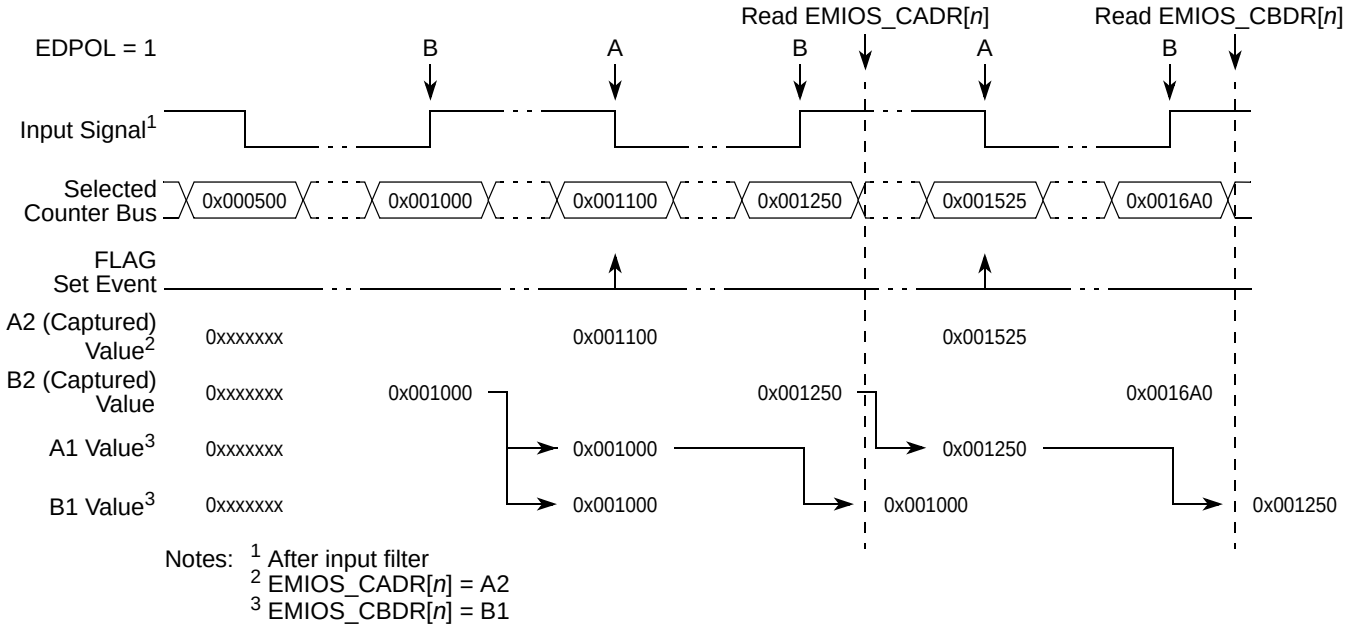


Figure 28-21. B1 and A1 Updates at EMIOS_CADR[*n*] and EMIOS_CBDR[*n*] Reads

Reading EMIOS_CADR[*n*] followed by EMIOS_CBDR[*n*] always provides coherent data. If no coherent data is required, the sequence of reads should be inverted, therefore EMIOS_CBDR[*n*] should be read prior to EMIOS_CADR[*n*] register. Even in this case B1 register updates are blocked after EMIOS_CADR[*n*] is read, therefore a second EMIOS_CBDR[*n*] is required to release the B1 register updates.

28.4.1.1.5 Input Period Measurement (IPM) Mode

The IPM mode (MODE = 000_0101) allows the measurement of the period of an input signal by capturing two consecutive rising edges or two consecutive falling edges. Successive input captures are done on consecutive edges of the same polarity. The edge polarity is defined by the EDPOL bit in the EMIOS_CCR[*n*] register.

When the first edge of selected polarity is detected, the selected time base is latched into the registers A2 and B2, and the data previously held in register B2 is transferred to register B1. On this first capture the FLAG line is not set and the values in registers B1 are meaningless. On the second and subsequent captures, the FLAG line is set and data in register B2 is transferred to register B1.

When the second edge of the same polarity is detected, the counter bus value is latched into registers A2 and B2, and the data previously held in register B2 is transferred to data register B1 and to register A1. The FLAG bit is set to indicate that the start and end points of a complete period have been captured. This sequence of events is repeated for each subsequent capture. Registers EMIOS_CADR[*n*] and EMIOS_CBDR[*n*] return the values in register A2 and B1, respectively.

To allow coherent data, reading EMIOS_CADR[*n*] forces A1 content be transferred to B1 register and disables transfers between B2 and B1. These transfers are disabled until the next read of the EMIOS_CBDR[*n*] register. Reading EMIOS_CBDR[*n*] register forces A1 content to be transferred to B1 and re-enables transfers from B2 to B1, to take effect at the next edge capture.

The input pulse period is calculated by subtracting the value in B1 from A2.

Figure 28-22 shows how the unified channel can be used for input period measurement.

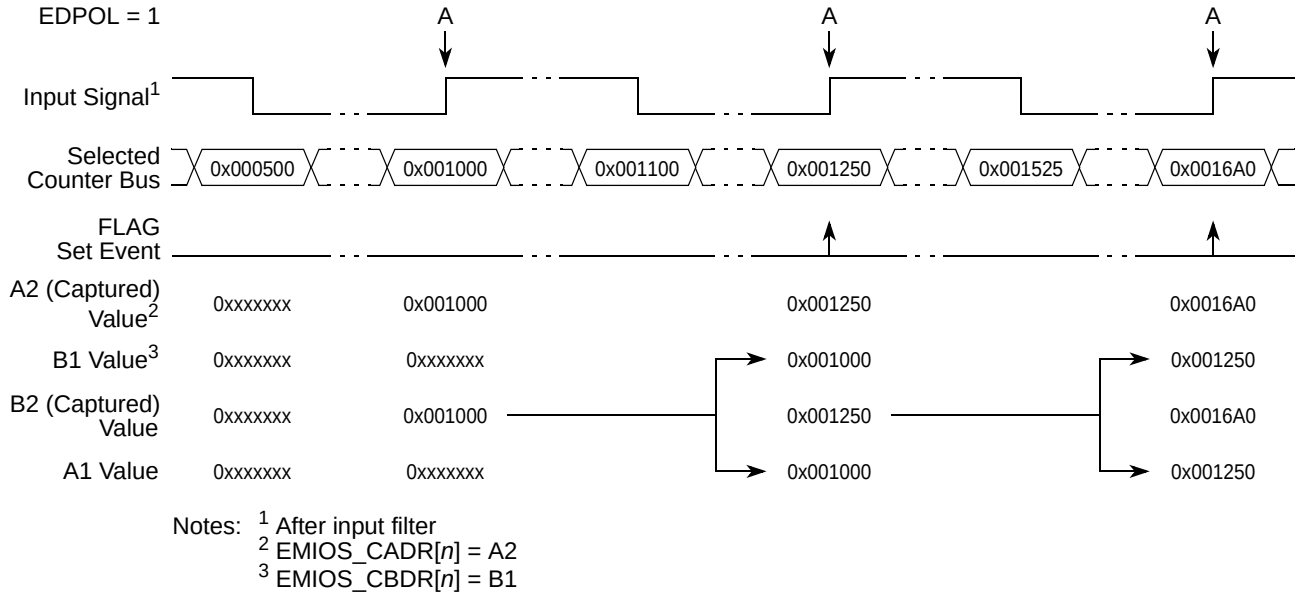


Figure 28-22. Input Period Measurement Example

Figure 28-23 shows the A1 and B1 register updates when EMIOS_CADR[*n*] and EMIOS_CBDR[*n*] read operations are performed. When EMIOS_CADR[*n*] read occurs, the content of A1 is transferred to B1 thus providing coherent data in A2 and B1 registers. Transfers from B2 to B1 are then blocked until EMIOS_CBDR[*n*] is read. After EMIOS_CBDR[*n*] is read, register A1 content is transferred to register B1 and the transfers from B2 to B1 are re-enabled to occur at the transfer edges, which is the leading edge in the Figure 28-23 example.

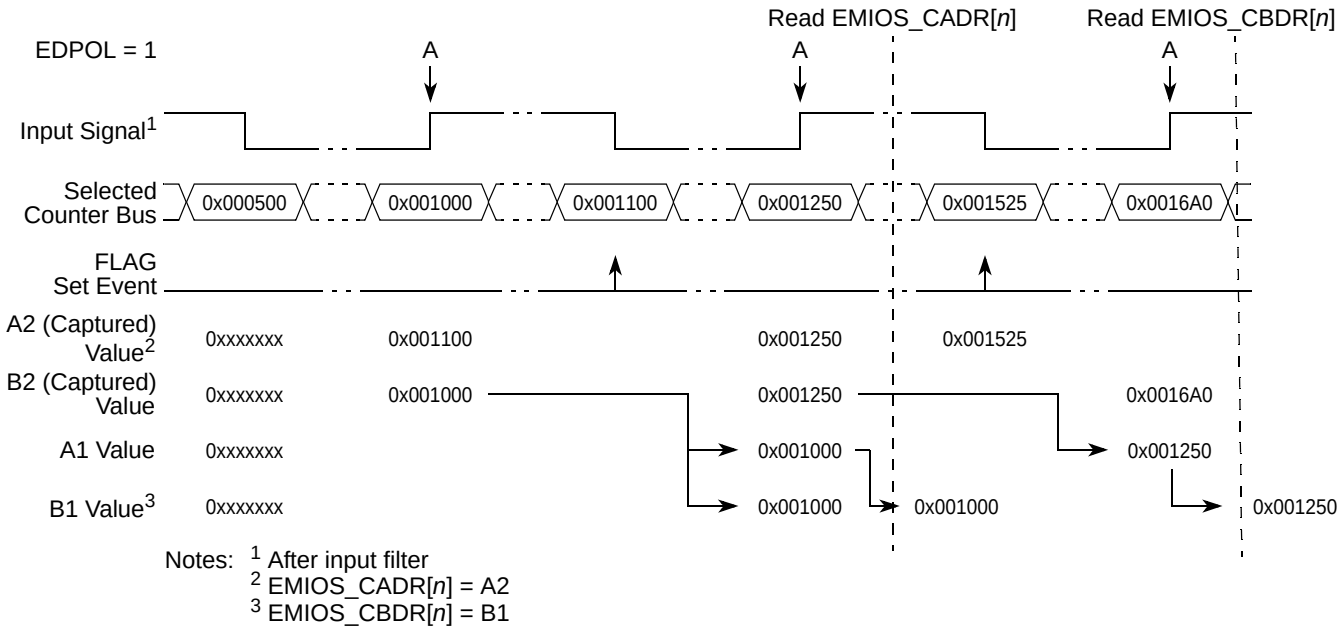


Figure 28-23. A1 and B1 Updates at EMIOS_CADR[*n*] and EMIOS_CBDR[*n*] Reads

28.4.1.1.6 Double Action Output Compare (DAOC) Mode

In the DAOC mode the leading and trailing edges of the variable pulse-width output are generated by matches occurring on comparators A and B, respectively.

When the DAOC mode is first selected (coming from GPIO mode) both comparators are disabled. Comparators A and B are enabled by updating registers A1 and B1 respectively and remain enabled until a match occurs on that comparator, when it is disabled again. In order to update registers A1 and B1, a write to A2 and B2 must occur and the OU[*n*] bit in EMIOS_OUDR must be cleared.

The output flip-flop is set to the value of EDPOL when a match occurs on comparator A and to the complement of EDPOL when a match occurs on comparator B.

MODE[6] controls if the FLAG is set on both matches or on the second match only (see Table 28-11 for details).

If subsequent enabled output compares occur on registers A1 and B1, pulses continue to be generated, regardless of the state of the FLAG bit.

At any time, the FORCMA and FORCMB bits allow the software to force the output flip-flop to the level corresponding to a comparison event in comparator A or B, respectively. The FLAG bit is not affected by these forced operations.

NOTE

If registers A1 and B1 are loaded with the same value, the unified channel behaves as if a single match on comparator B had occurred, i.e., the output pin is set to the complement of EDPOL bit and the FLAG bit is set.

Figure 28-24 and Figure 28-25 show how the unified channel can be used to generate a single output pulse with FLAG bit being set on the second match or on both matches, respectively.

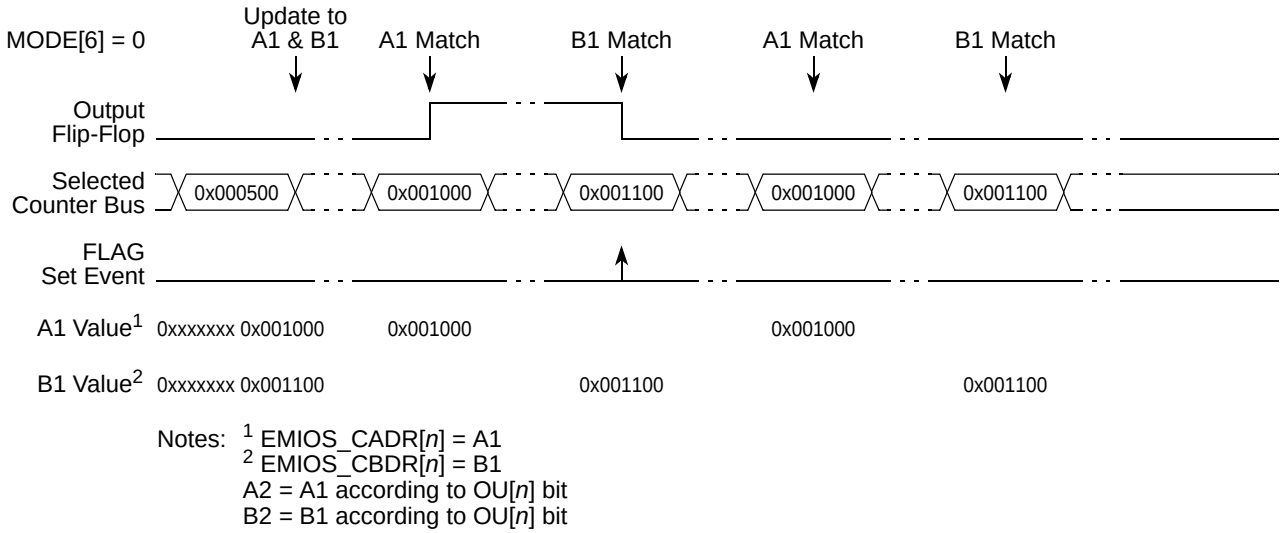
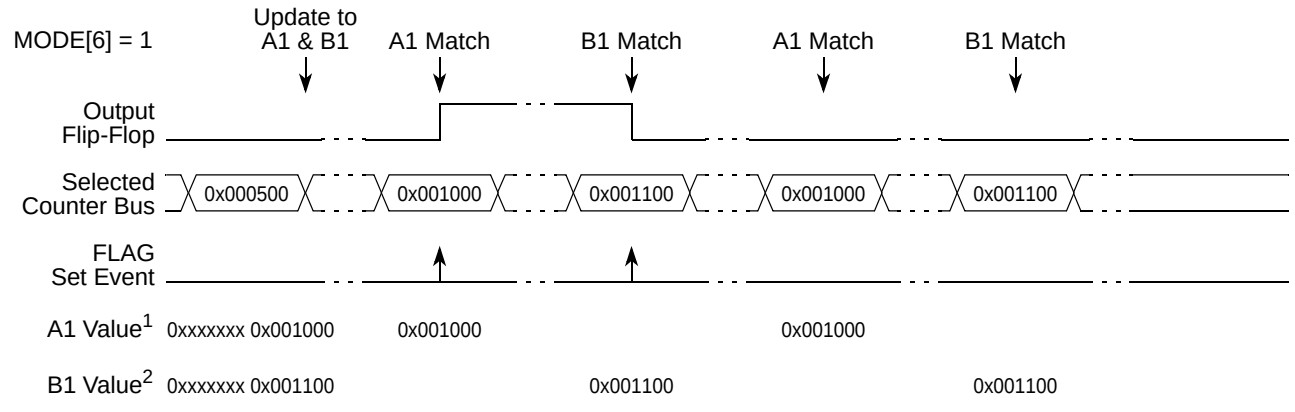
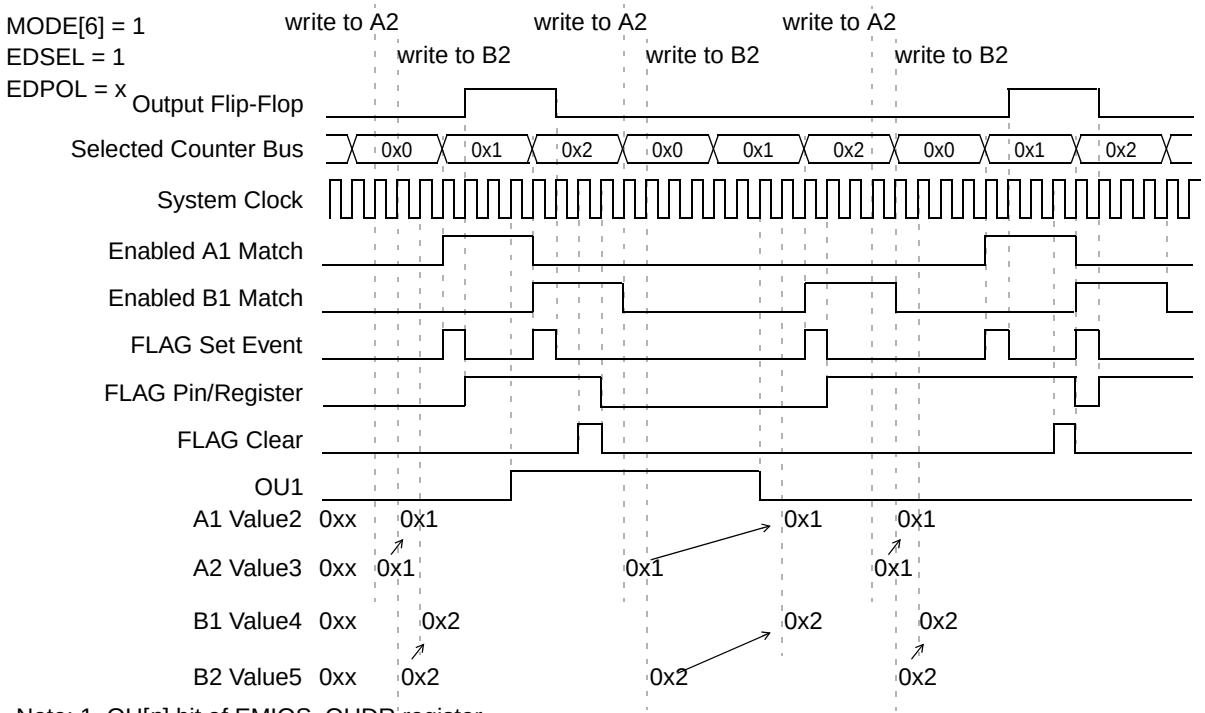


Figure 28-24. Double Action Output Compare with FLAG Set on the Second Match



Notes: [1] EMIOS_CADR[*n*] = A1
[2] EMIOS_CBDR[*n*] = B1
A2 = A1 according to OU[*n*] bit
B2 = B1 according to OU[*n*] bit

Figure 28-25. Double Action Output Compare with FLAG Set on Both Matches



Note: 1. OU[*n*] bit of EMIOS_OUDR register
2. EMIOS_CADR[*n*] = A1 (when reading)
3. EMIOS_CADR[*n*] = A2 (when writing)
4. EMIOS_CBDR[*n*] = B1 (when reading)
5. EMIOS_CBDR[*n*] = B2 (when writing)

Figure 28-26. DAOC with Transfer Disabling Example

28.4.1.1.7 Pulse/Edge Accumulation (PEA) Mode

The PEA mode returns the time taken to detect a desired number of input events. MODE[6] bit selects between continuous or single shot operation.

After writing to register A1, the internal counter is cleared on the first input event, ready to start counting input events and the selected timebase is latched into register B2. On the match between the internal counter and register A1, a counter bus capture is triggered to register A2 and B2. The data previously held in register B2 is transferred to register B1 and the FLAG bit is set to indicate that an event has occurred.

The desired time interval can be determined by subtracting register B1 from A2. Registers EMIOS_CADR[*n*] and EMIOS_CBDR[*n*] return the values in register A2 and B1, respectively.

As part of the coherency mechanism, reading EMIOS_CADR[*n*] disables transfers from B2 to B1. These transfers are disabled until the next read of the EMIOS_CBDR[*n*] register. Reading the EMIOS_CBDR[*n*] register re-enables transfers from B2 to B1, to take effect at the next transfer event, as previously described.[1]

In order to have coherent data in continuous mode of operation the following steps should be performed, assuming FLAG is initially cleared:

1. Wait for FLAG assertion.
2. Read EMIOS_CADR[*n*] register.
3. Read EMIOS_CBDR[*n*] register.
4. Clear FLAG bit.
5. Return to step #1.

Accumulation cycles may be lost if the read is not performed in a timely manner. Whenever the Overrun bit is asserted it means that one or more cycles have been lost.

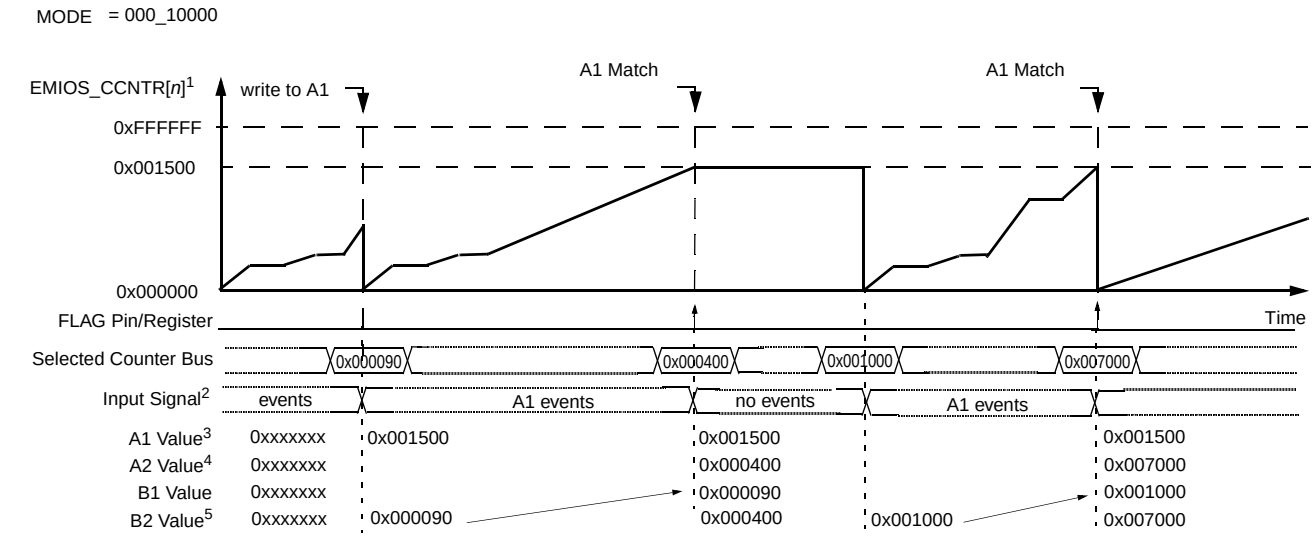
Triggering of the counter clock (input event) is done by a rising or falling edge or both edges on the input pin. The polarity of the triggering edge is selected by the EDSEL and EDPOL bits in EMIOS_CCR[*n*] register.

For continuous operation mode (MODE[0] cleared, MODE[0:6] = 000_1000), the counter is cleared on the next input event after a FLAG generation and continues to operate as previously described.

For single shot operation (MODE[0] set, MODE[0:6] = 000_1001), the counter is not cleared or incremented after a FLAG generation until a new writing operation to register A is performed.

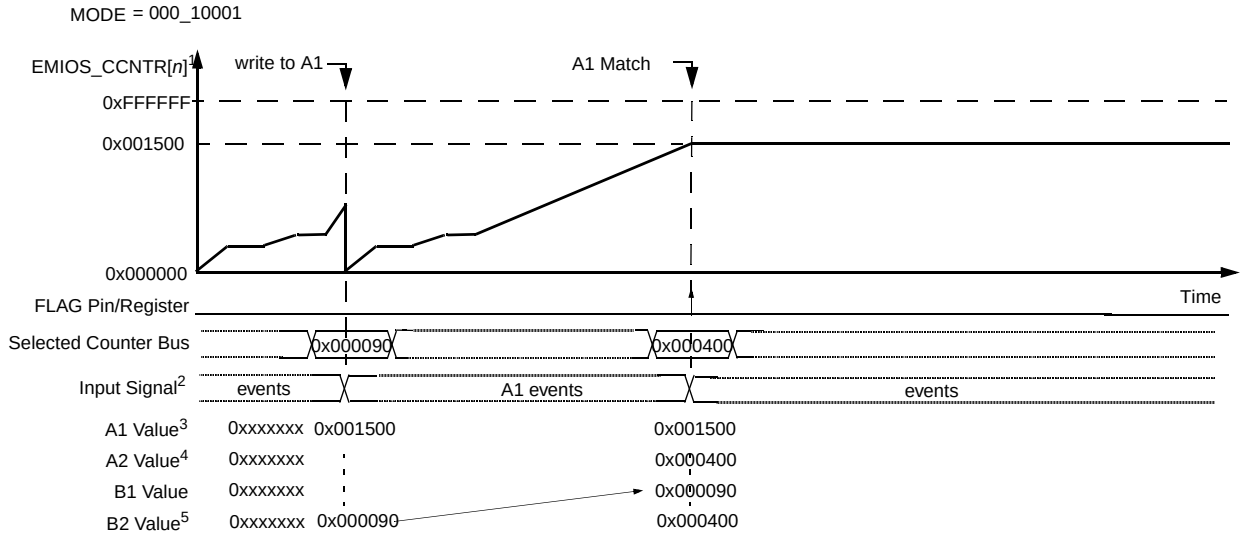
Figure 28-27 and Figure 28-28 show how the Unified Channel can be used for continuous and single shot pulse/edge accumulation mode.

1. If B1 was not updated due to B2 to B1 transfer being disabled after reading register EMIOS_CADR[*n*], further EMIOS_CADR[*n*] and EMIOS_CBDR[*n*] reads will not return coherent data until a new bus capture is triggered to registers A2 and B2. This capture event is indicated by the channel FLAG being asserted. If enabled, the FLAG also generates an interrupt.



Notes: 1. Cleared on the first input event after writing to register A1
2. After input filter
3. EMIOS_CADR[*n*] = A1 (when writing)
4. EMIOS_CADR[*n*] = A2 (when reading)
5. EMIOS_CBDR[*n*] = B1

Figure 28-27. Pulse/Edge Accumulation Continuous Mode Example



Notes: 1. Cleared on the first input event after writing to register A1
2. After input filter
3. EMIOS_CADR[*n*] = A1 (when writing)
4. EMIOS_CADR[*n*] = A2 (when reading)
5. EMIOS_CBDR[*n*] = B1

Figure 28-28. Pulse/Edge Accumulation Single-Shot Mode Example

28.4.1.1.8 Pulse/Edge Counting (PEC) Mode

The PEC mode returns the amount of pulses or edges detected on the input for a desired time window. MODE[6] bit selects between continuous or single shot operation.

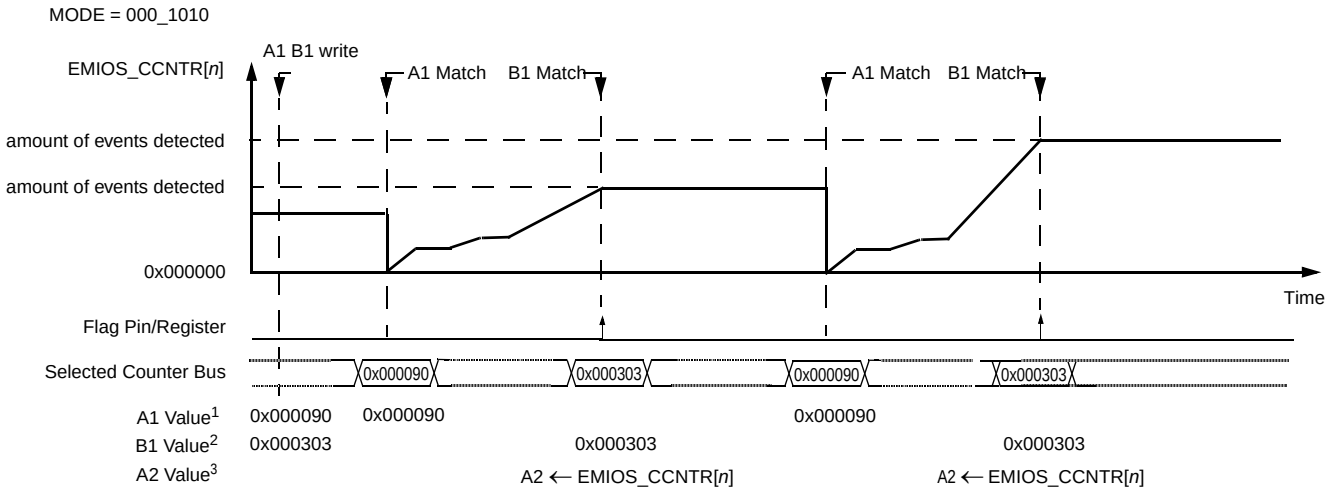
Triggering of the internal counter is done by a rising or falling edge or both edges on the input signal. The polarity and the triggering edge is selected by EDSEL and EDPOL bits in EMIOS_CCR[*n*] register.

Register A1 holds the start time and register B1 holds the stop time for the time window. After writing to register A1, when a match occur between comparator A and the selected timebase, the internal counter is cleared and it is ready to start counting input events. When the time base matches comparator B, the internal counter is disabled and its content is transferred to register A2. At the same time the FLAG bit is set. Reading registers EMIOS_CCNTR[*n*] or A2 returns the amount of detected pulses.

For continuous operation (MODE[6] cleared, MODE[0:6] = 000_1010), the next match between comparator A and the selected time base clears the internal counter and counting is enabled again. In order to guarantee coherent measurements when reading EMIOS_CCNTR[*n*] after the FLAG is set, the software must check if the time base value is out of the time interval defined by registers A1 and B1. Alternatively register A2 always holds the latest available measurement providing coherent data at any time after the first FLAG had occurred. This register is addressed by the alternate address EMIOS_ALTA[*n*].

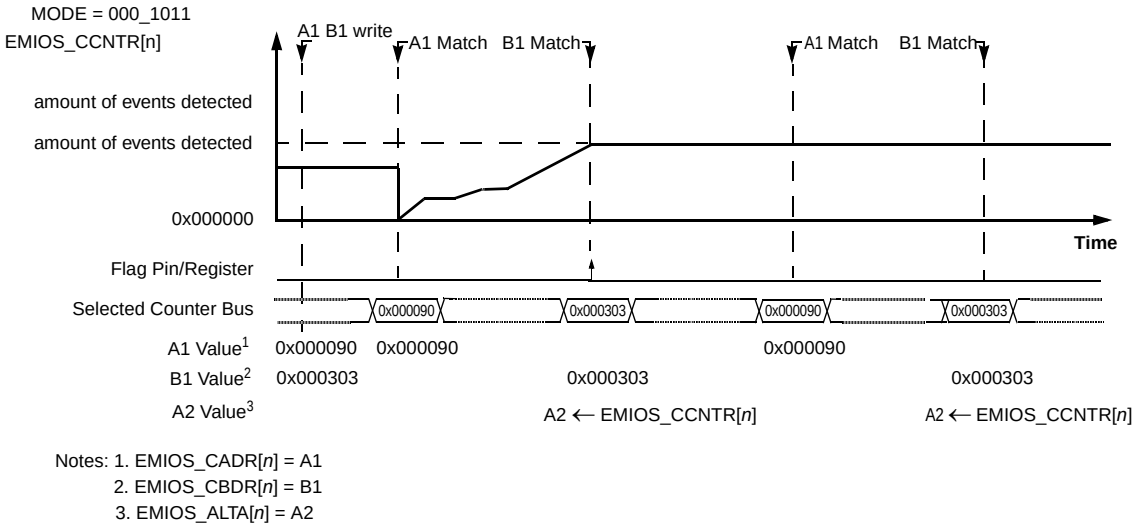
For single shot operation (MODE[6] set, MODE[0:6] = 000_1011), the next match between comparator A and the selected time base has no effect, until a new write to register A is performed. The EMIOS_CCNTR content is also transferred to register A2 when a match in the B comparator occurs.

Figure 28-29 and Figure 28-30 show how the Unified Channel can be used for continuous or single shot pulse/edge counting mode.



Notes: 1. EMIOS_CADR[*n*] = A1
2. EMIOS_CBDR[*n*] = B1
3. EMIOS_ALTA[*n*] = A2

Figure 28-29. Pulse/Edge Counting Continuous Mode Example



Notes: 1. EMIOS_CADR[*n*] = A1
2. EMIOS_CBDR[*n*] = B1
3. EMIOS_ALTA[*n*] = A2

Figure 28-30. Pulse/Edge Counting Single-Shot Mode Example

28.4.1.1.9 Quadrature Decode (QDEC) Mode

Quadrature decode mode uses UC[*n*] operating in QDEC mode and the input programmable filter (IPF) from UC[*n* – 1]. Note that UC[*n* – 1] can be configured, at the same time, to an operation mode that does not use I/O pins, such as MC mode (modulus counter). The connection among the UCs is circular, i.e., when UC[0] is running in QDEC mode, the input programmable filter from UC[23] is being used.

This mode generates a FLAG every time the internal counter matches A1 register. The internal counter is automatically selected and is not cleared when entering this mode.

MODE[0] bit selects which type of encoder is used: *count & direction* encoder or *phase_A & phase_B* encoder.

When operating with *count & direction* encoder (MODE[6] cleared), UC[*n*] input pin must be connected to the *direction* signal and UC[*n* – 1] input pin must be connected to the count signal of the quadrature encoder. UC[*n*] EDPOL bit selects count direction according to *direction* signal and UC[*n* – 1] EDPOL bit selects if the internal counter is clocked by the rising or falling edge of the *count* signal.

When operating with *phase_A & phase_B* encoder (MODE[6] set), UC[*n*] input pin must be connected to the *phase_A* signal and UC[*n* – 1] input pin must be connected to the *phase_B* signal of the quadrature encoder. EDPOL bit selects the count direction according to the phase difference between *phase_A & phase_B* signals.

Figure 28-31 and Figure 28-32 show two Unified Channels configured to quadrature decode mode for *count & direction* encoder and *phase_A & phase_B* encoders, respectively.

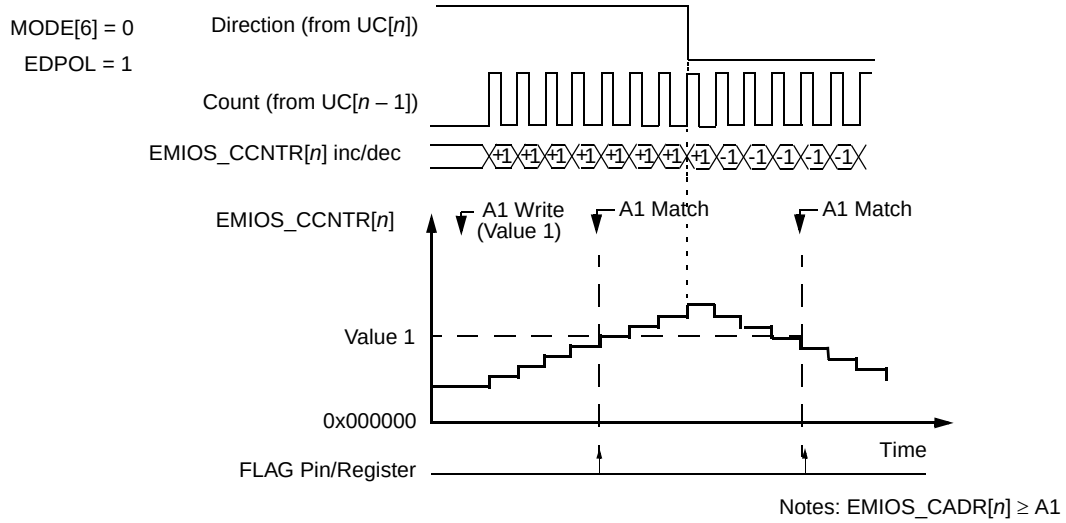


Figure 28-31. Quadrature Decode Mode Example with *Count & Direction* Encoder

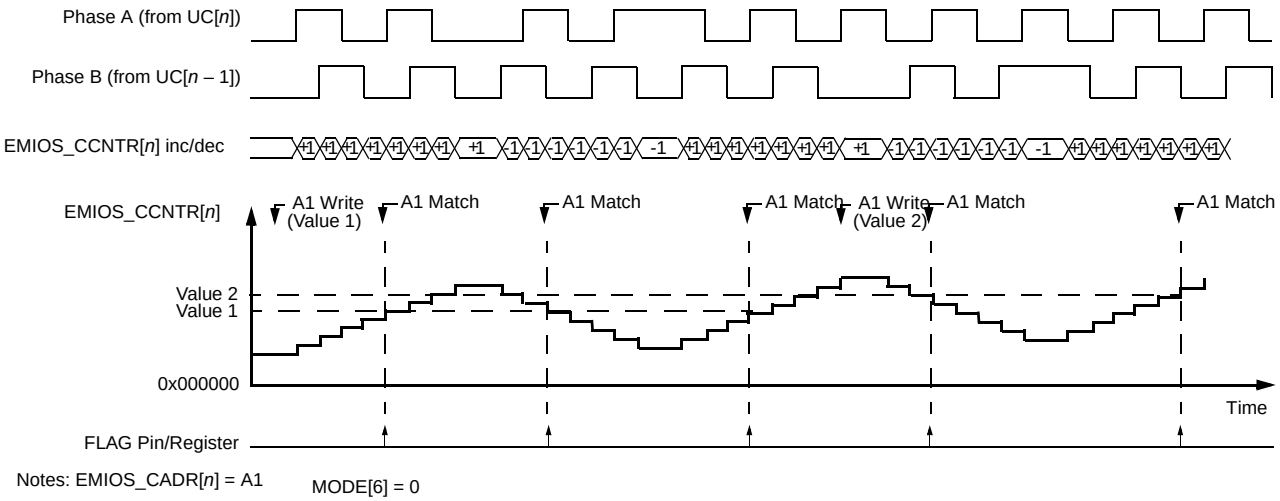


Figure 28-32. Quadrature Decode Mode Example with *Phase_A & Phase_B* Encoder

28.4.1.1.10 Modulus Counter (MC) Mode

The MC mode can be used to provide a time base for a counter bus or as a general purpose timer.

MODE[6] bit selects internal or external clock source when cleared or set, respectively. When external clock is selected, the input signal pin is used as the source and the triggering polarity edge is selected by the EDPOL and EDSEL in the EMIOS_CCR[*n*] register.

The internal counter counts up from the current value until it matches the value in register A1. Register B1 is cleared and is not accessible to the MCU. The MODE[4] bit selects up mode or up/down mode, when cleared or set, respectively.

When in up count mode, a match between the internal counter and register A1 sets the FLAG and clears the internal counter. The timing of those events varies according to the MC mode setup as follows:

- Internal counter clearing on match start (MODE[0:6] = 001_000b)
  - External clock is selected if MODE[6] is set. In this case the internal counter clears as soon as the match signal occurs. The channel FLAG is set at the same time the match occurs. Note that by having the internal counter cleared as soon as the match occurs and incremented at the next input event a shorter zero count is generated. See Figure 28-57 and Figure 28-59.
  - Internal clock source is selected if MODE[6] is cleared. In this case the counter clears as soon as the match signal occurs. The channel FLAG is set at the same time the match occurs. At the next prescaler tick after the match the internal counter remains at zero and only resumes counting on the following tick. See Figure 28-57 and Figure 28-60.
- Internal counter clearing on match end (MODE[0:6] = 001_001b)
  - External clock is selected if MODE[6] is set. In this case the internal counter clears when the match signal is asserted and the input event occurs. The channel FLAG is set at the same time the counter is cleared. See Figure 28-57 and Figure 28-61.
  - Internal clock source is selected if MODE[6] is cleared. In this case the internal counter clears when the match signal is asserted and the prescaler tick occurs. The channel FLAG is set at the same time the counter is cleared. See Figure 28-57 and Figure 28-61.

NOTE

If internal clock source is selected and the prescaler of the internal counter is set to 1, the MC mode behaves the same way even in Clear on Match Start or Clear on Match End sub-modes.

When in up/down count mode (MODE[0:6] = 001_01bb), a match between the internal counter and register A1 sets the FLAG and changes the counter direction from increment to decrement. A match between register B1 and the internal counter changes the counter direction from decrement to increment and sets the FLAG only if MODE[5] bit is set.

Only values other than 0x00_0000 must be written into register A. Loading 0x00_0000 leads to unpredictable results.

Updates on register A or the counter in MC mode may cause loss of match in the current cycle if the transfer occurs near the match. In this case, the counter may roll over and resume operation in the next cycle.

Figure 28-33 and Figure 28-34 show how the Unified Channel can be used as a modulus counter in up mode and up/down mode, respectively.

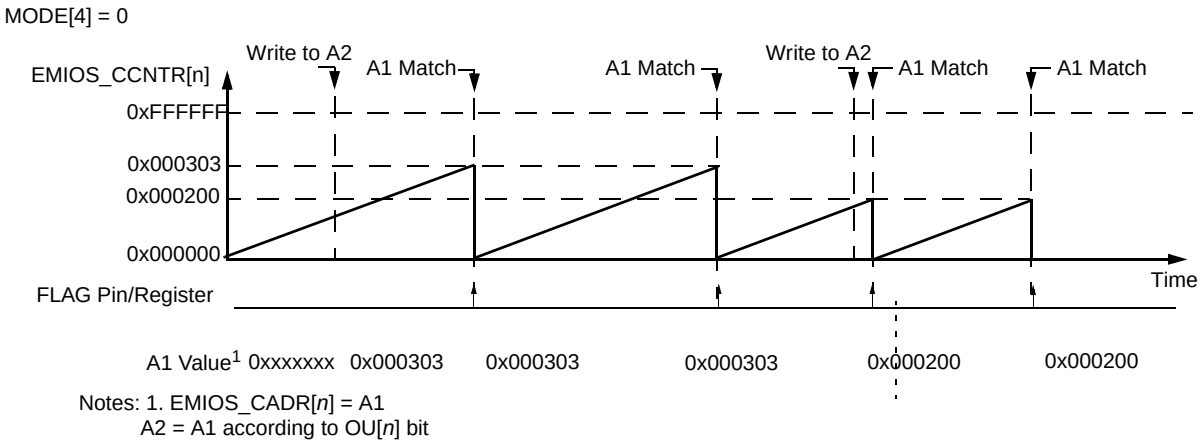


Figure 28-33. Modulus Counter Up Mode Example

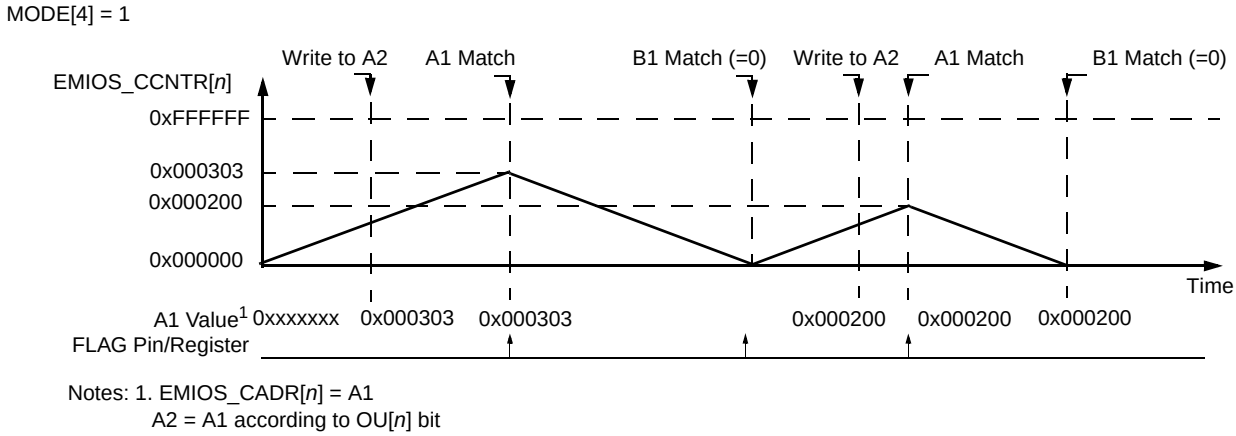


Figure 28-34. Modulus Counter Up/Down Mode Example

28.4.1.1.11 Modulus Counter Buffered (MCB) Mode

The MCB mode provides a time base that can be shared with other channels through the internal counter buses. Register A1 is double buffered, thus allowing smooth transitions between cycles when changing A2 register value. Register A1 is updated at the cycle boundary, which is defined as when the internal counter reaches the value one. The internal counter values are within a range from 0x00_0001 to the value of register A1 in MCB mode. The internal counter must not reach 0x00_0000 as a consequence of a rollover.To avoid this, the user must start MCB only if the value stored in the internal counter is less than the value stored in the EMIOS_CADR[*n*] register.

MODE[6] bit selects the internal clock source if set to 0 or external, if set to 1. When the external clock is selected the input channel pin is used as the channel clock source. The active edge of this clock is defined by EDPOL and EDSEL bits in the EMIOS_CCR channel register.

When entering in MCB mode, if the up counter is selected by MODE[4] = 0, the internal counter starts counting from its current value to up direction until A1 match occurs. On the next system clock cycle after the A1 match occurs, the internal counter is set to one. If up/down counter is selected by setting

MODE[4] = 1, the counter changes direction at the A1 match and counts down until it reaches the value one. After it has reached one, it is set to count in up direction again. Register B1 is set to one at mode entering and cannot be changed while this mode is selected. B1 register is used to generate a match to set the internal counter in up-count direction if up/down mode is selected.

The MCB mode counts between one and A1 register value. Only values greater than 0x00_0001 are allowed to be written at A1 register. Loading values other than those leads to unpredictable results. The counter cycle period is equal to A1 value in up counter mode. If in up/down counter mode the period is defined by the expression: (2*A1) – 2.

Figure 28-35 shows the counter cycle for several A1 values. Register A1 is loaded with A2 register value at the cycle boundary. Any value written to A2 register within cycle (*n*) is updated to A1 at the next cycle boundary and therefore is used on cycle (*n* + 1). The cycle boundary between cycle (*n*) and cycle (*n* + 1) is defined as the first system clock cycle of cycle (*n* + 1). The flags are generated as soon as A1 match had occurred.

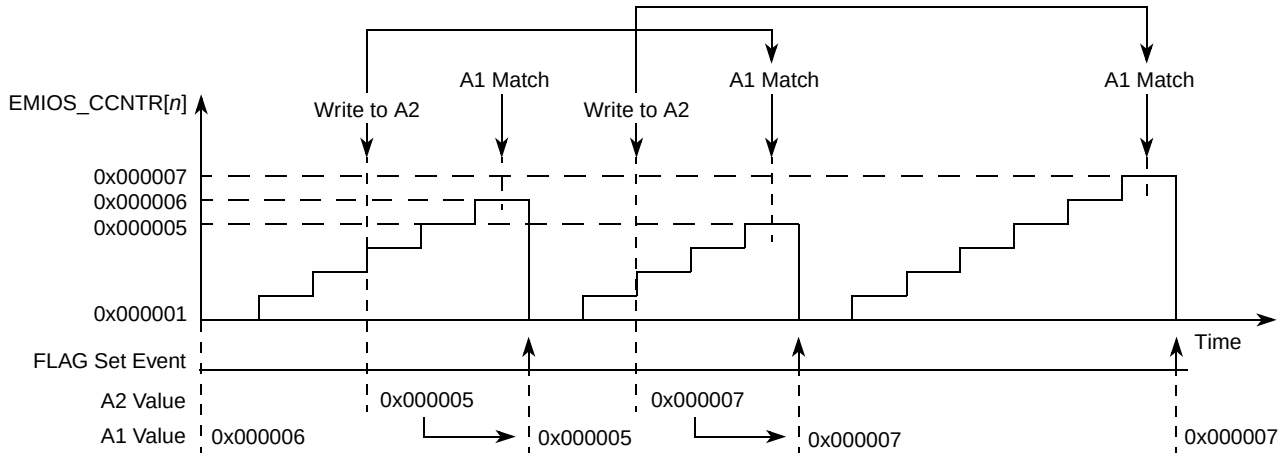


Figure 28-35. Modulus Counter Buffered (MCB) Up Count Mode

Figure 28-36 shows the MCB in up/down counter mode. Register A1 is updated at the cycle boundary. If A2 is written in cycle (*n*), this new value is used in cycle (*n* + 1) for A1 match.

Flags are generated at A1 match only if MODE[5] is 0. If MODE[5] is set to 1, flags are also generated at the cycle boundary.

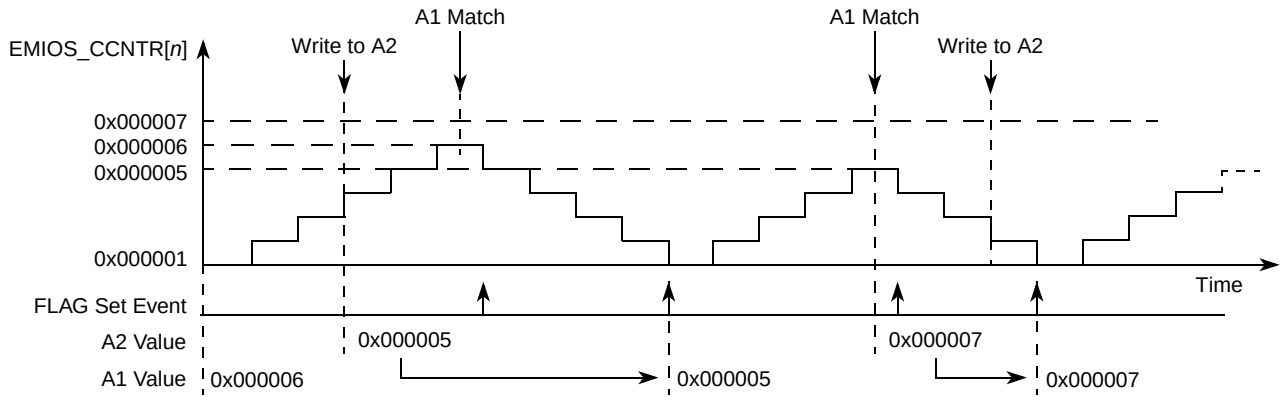


Figure 28-36. Modulus Counter Buffered (MCB) Up/Down Mode

Figure 28-37 shows the A1 register update process in up counter mode. The A1 load signal is generated based on the detection of the internal counter reaching one and has the duration of one system clock cycle. During the load pulse, A1 still holds its previous value. It is updated at the second system clock cycle only.

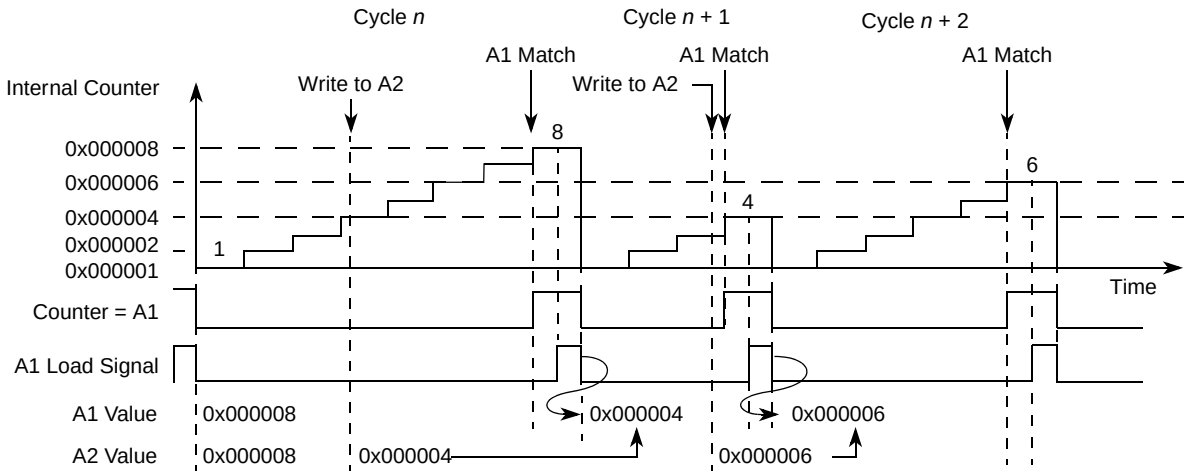


Figure 28-37. MCB Mode A1 Register Update in Up Counter Mode

Figure 28-38 shows the A1 register update in up/down counter mode. Note that A2 can be written at any time within cycle (*n*) in order to be used in cycle (*n* + 1). Thus A1 receives this new value at the next cycle boundary. The update disable bits (OU[*n*] in EMIOS_OUDR) can be used to disable the update of A1 register.

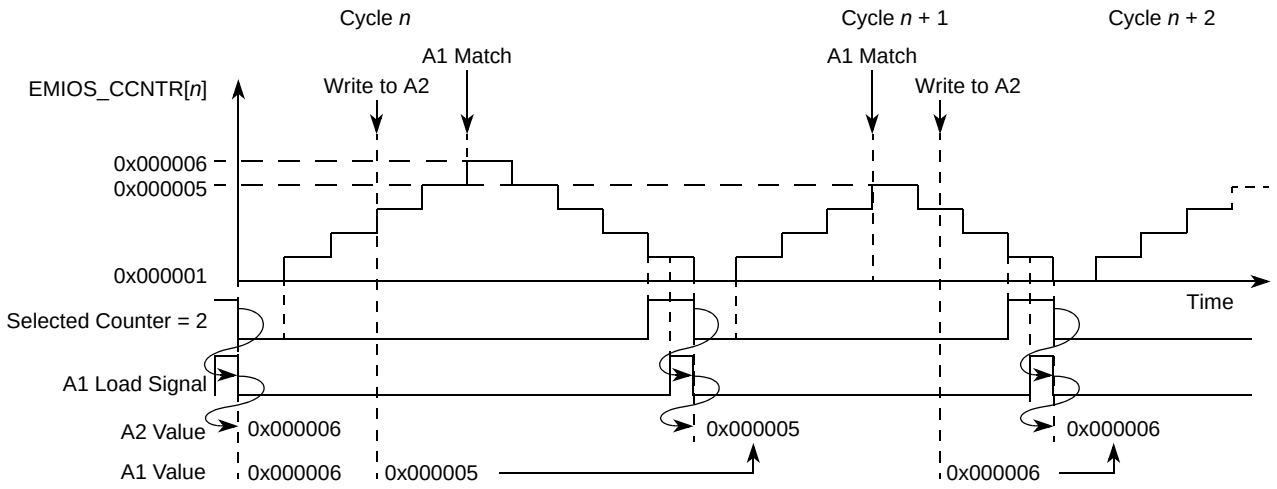


Figure 28-38. MCB Mode A1 Register Update in Up/Down Counter Mode

28.4.1.1.12 Pulse-Width and Frequency Modulation Buffered (OPWFMB) Mode

This mode provides waveforms with variable duty cycle and frequency. The internal channel counter is automatically selected as the time base when this mode is selected. A1 register indicates the duty cycle and B1 register the frequency. Both A1 and B1 registers are double buffered to allow smooth signal generation when changing the registers values. It supports 0% and 100% duty cycles.

To provide smooth and consistent channel operation, this mode differs substantially from the OPWFM mode. The main differences reside in the A1 and B1 registers update, on the delay from the A1 match to the output pin transition, and on the range of the internal counter values, which start from 1 and go as high as the value in the B1 register. The internal counter must not reach 0x00_0000 as consequence of a rollover. To avoid this, the user must start OPWFMB only if the value stored at internal counter is fewer than the value that EMIOS_CBDR register stores.

When a match on comparator A occurs the output register is set to the value of EDPOL. When a match on comparator B occurs the output register is set to the complement of EDPOL. B1 match also causes the internal counter to transition to 1, thus restarting the counter cycle.

Only values greater than 0x00_0001 are allowed to be written to B1 register. Loading values other than those leads to unpredictable results.

Figure 28-39 shows the operation of the OPWFMB mode regarding output pin transitions and A1/B1 registers match events. The output pin transition occurs when the A1 or B1 match signal is deasserted, which is indicated by the A1 match negedge detection signal. If register A1 is set to 0x00_0004, the output pin transitions four counter periods after the cycle has started, plus one system clock cycle. In the example shown in Figure 28-39 the internal counter prescaler is set to two.

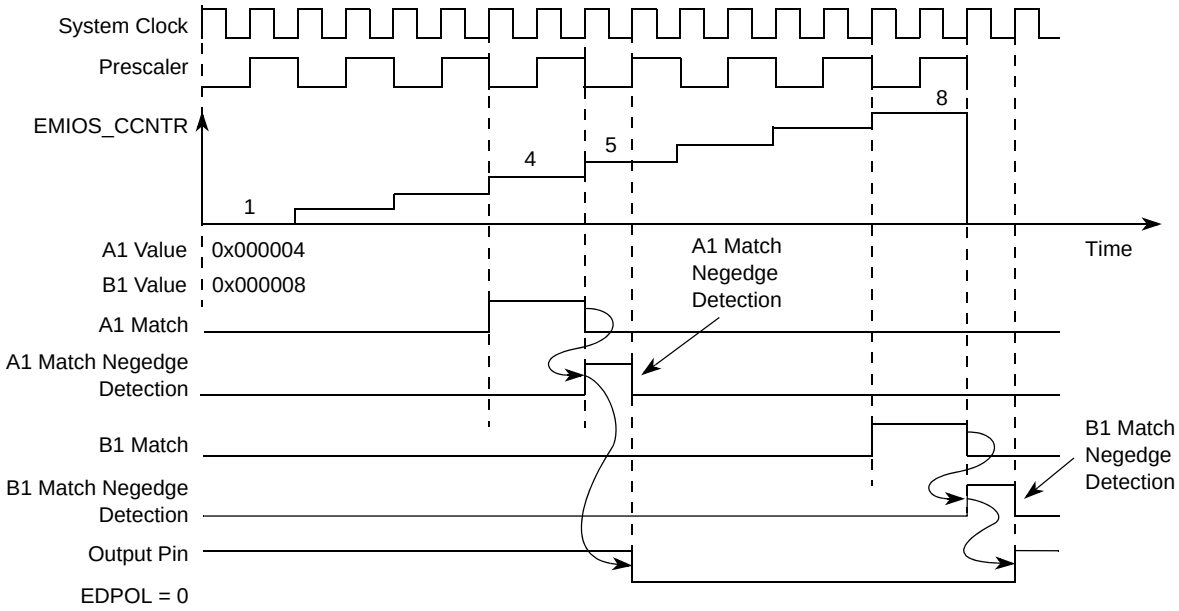


Figure 28-39. OPWFMB A1 and B1 Match to Output Register Delay

Figure 28-40 shows the generated output signal if A1 is set to zero. Because the counter does not reach zero in this mode, the channel internal logic infers a match as if A1 = 0x00_0001 with the difference that in this case, the posedge of the match signal is used to trigger the output pin transition instead of the negedge used when A1 = 0x00_0001. A1 posedge match signal from cycle (*n* + 1) occurs at the same time as B1 negedge match signal from cycle (*n*). This allows using the A1 posedge match to mask the B1 negedge match when they occur at the same time. The result is that no transition occurs on the output flip-flop and a 0% duty cycle is generated.

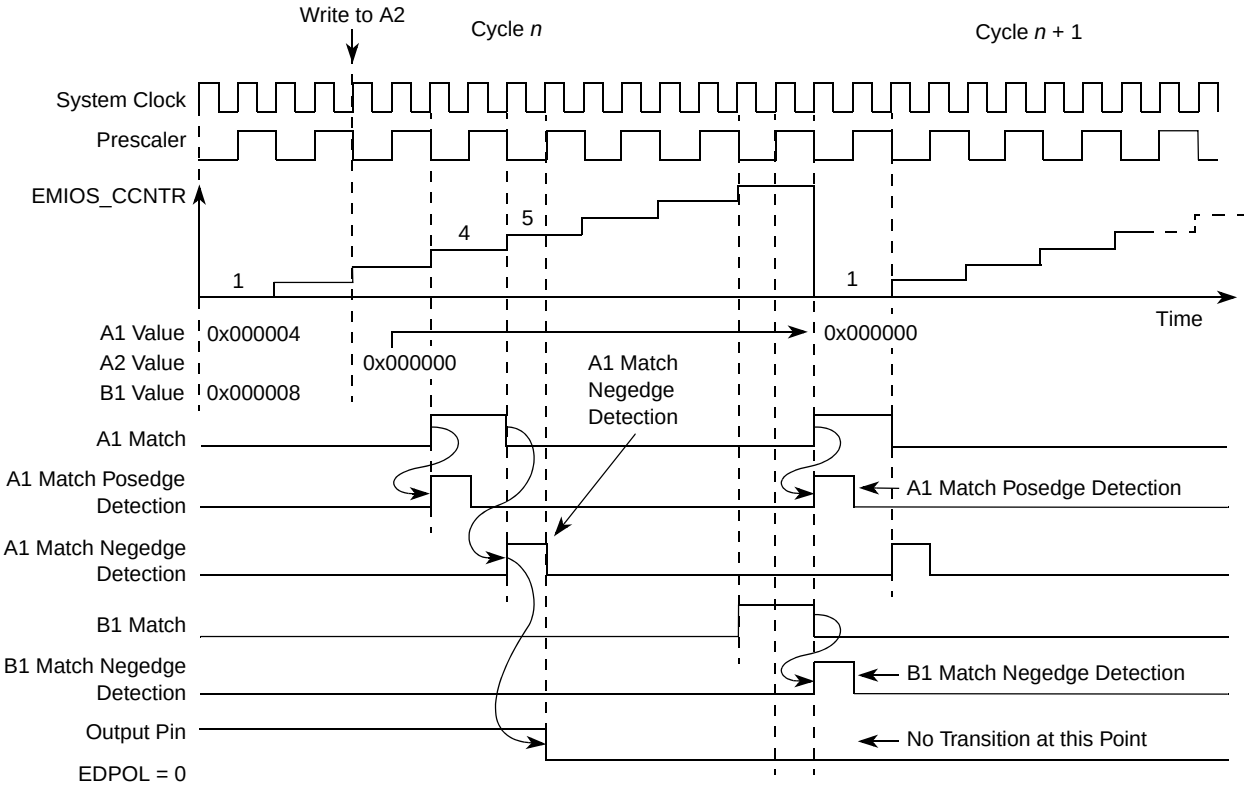


Figure 28-40. OPWFMB Mode with A1 = 0 (0% duty cycle)

Figure 28-41 shows the timing for the A1 and B1 registers load. The A1 and B1 load use the same signal that is generated based on the selected counter reaching the value one, or EMIOS_CCNTR[*n*] = 1. This event is defined as the cycle boundary. The load signal pulse has the duration of one system clock cycle and occurs at the first system clock period of every counter cycle. If A2 and B2 are written within cycle (*n*), their values are loaded into A1 and B1, respectively, at the first clock of cycle (*n* + 1) and the new values are used for matches at cycle (*n* + 1). The update disable bits (OU[*n*] in EMIOS_OUDR) can be used to control the update of these registers, thus allowing to delay the A1 and B1 registers update for synchronization purposes.

In Figure 28-41, it is assumed that the channel and global prescalers are set to one, meaning that the channel internal counter transition at every system clock cycle. FLAGs can be generated only on B1 matches when MODE[5] is cleared, or on either A1 or B1 matches when MODE[5] is set. Because B1 FLAG occurs at the cycle boundary, this flag can be used to indicate that A2 or B2 data written on cycle (*n*) were loaded to A1 or B1, respectively, thus generating matches in cycle (*n* + 1).

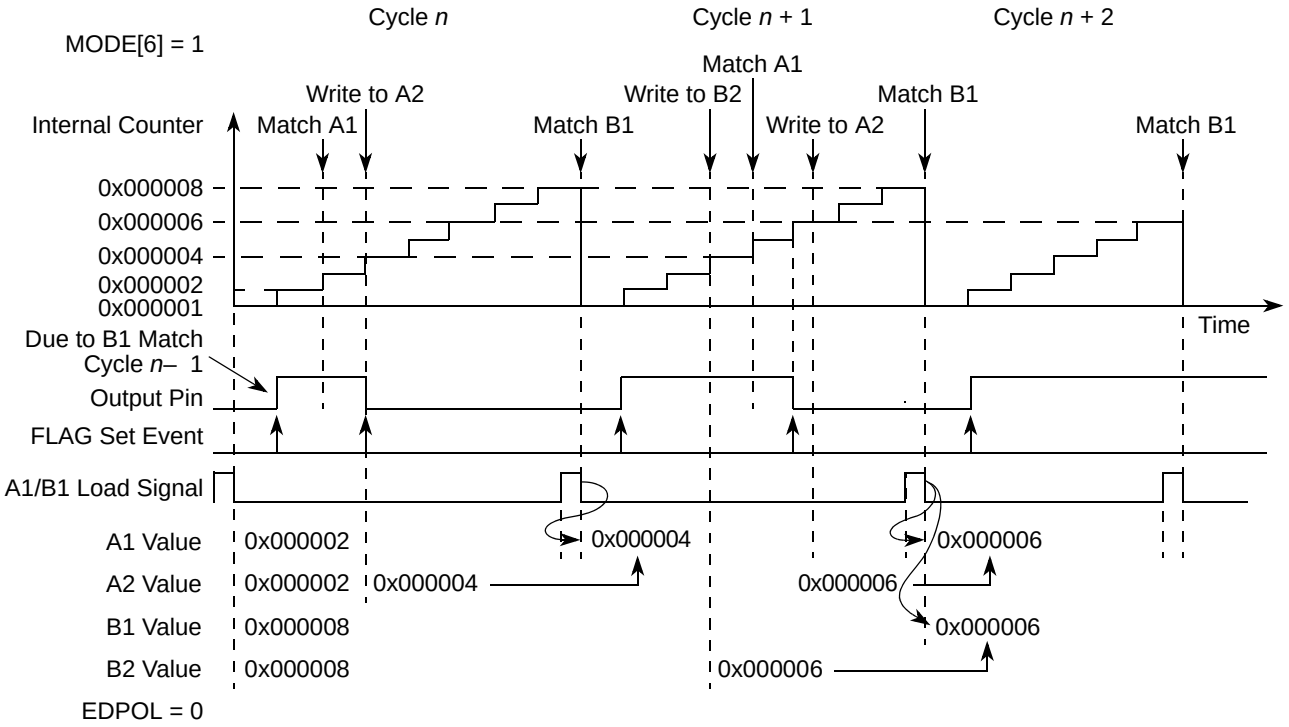


Figure 28-41. OPWFMB A1 and B1 Registers Update and Flags

Figure 28-42 shows the operation of the output disable feature in OPWFMB mode. The output disable forces the channel output flip-flop to EDPOL bit value. This functionality targets applications that use active high signals and a high to low transition at A1 match. In this case EDPOL should be set to 0.

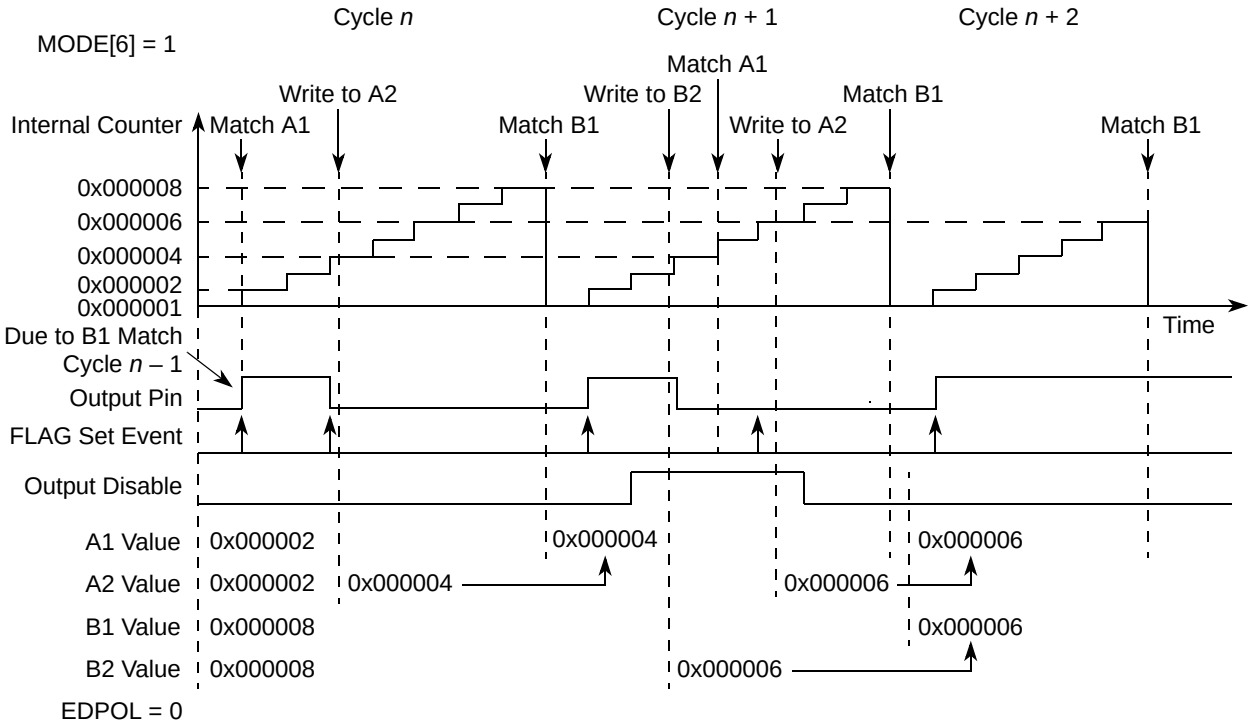


Figure 28-42. OPWFMB Mode with Active Output Disable

The output disable has a synchronous operation, meaning that the assertion of the output disable input pin causes the channel output flip-flop to transition to EDPOL at the next system clock cycle. If the output disable input is deasserted the output pin transition at the following A1 or B1 match.

In Figure 28-42 it is assumed that the output disable input is enabled and selected for the channel. Refer to Section 28.3.2.8, eMIOS200 Control Register (EMIOS_CCR[n]), for a description of how the ODIS and ODISSL bits enable and select the output disable inputs.

The FORCMA and FORCMB bits allow the software to force the output flip-flop to the level corresponding to a match on comparators A or B respectively. Similar to a B1 match FORCMB sets the internal counter to 0x00_0001. The FLAG bit is not set by the FORCMA or FORCMB bits being asserted.

Figure 28-43 shows the generation of 100% and 0% duty cycle signals. It is assumed EDPOL = 0 and the resultant prescaler value is 1. Initially, A1 = 0x00_0008 and B1 = 0x00_0008. In this case, the B1 match has precedence over the A1 match, thus the output flip-flop is set to the complement of EDPOL bit. This cycle corresponds to a 100% duty cycle signal. The same output signal can be generated for any A1 value greater or equal to B1.

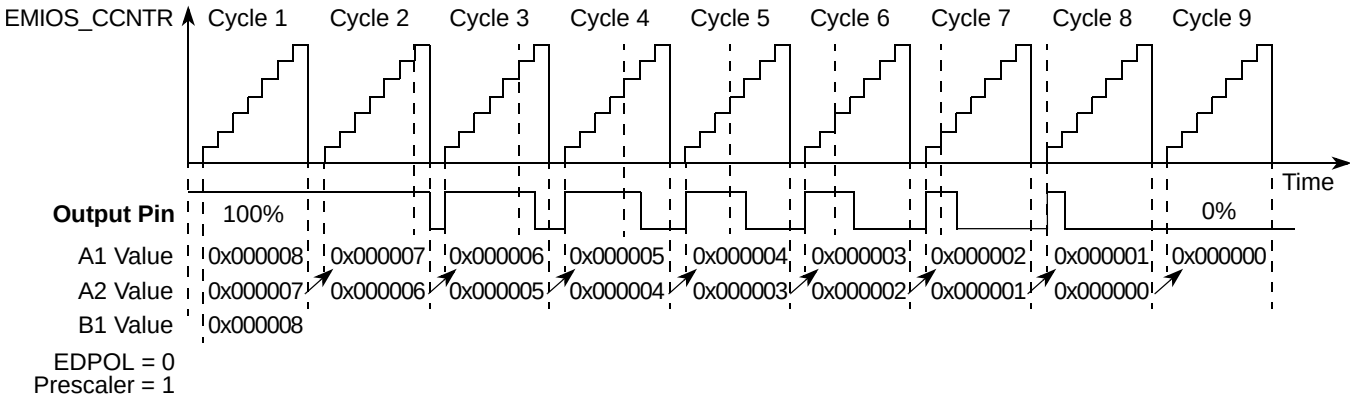


Figure 28-43. OPWFMB Mode from 100% to 0% Duty Cycle

A 0% duty cycle signal is generated if A1 = 0 as shown in cycle 9 in Figure 28-43. In this case, the B1 = 0x00_0008 match from cycle 8 occurs at the same time as the A1 = 0x00_0000 match from cycle 9. Refer to Figure 28-40 for a description of the A1 and B1 match generation. In this case, the A1 match has precedence over the B1 match and the output signal transitions to EDPOL.

28.4.1.1.13 Center-Aligned Output PWM Buffered with Dead-Time (OPWMCB) Mode

This operation mode generates a center-aligned PWM with dead-time insertion to the leading or trailing edge. A1 and B1 registers are double buffered to allow smooth output signal generation when changing A2 or B2 registers values.

The selected counter bus must be running in up/down counter mode, as shown in Figure 28-36. The time base selected for a channel configured to OPWMCB mode should be a channel configured to MCB mode. The BSL bits select the time base. The time base must start at 0x00_0001 and upward not prior to OPWMCB mode is active. Register A1 contains the ideal duty cycle for the PWM signal and is compared with the selected time base. Register B1 contains the dead-time value and is compared against the internal counter. For a leading edge dead-time insertion, the output PWM duty cycle is equal to the difference between register A1 and register B1, and for a trailing edge dead-time insertion, the output PWM duty cycle is equal to the sum of register A1 and register B1. MODE[6] bit selects between trailing and leading dead-time insertion, respectively.

NOTE

The internal prescaler of the OPWMCB channel must be set to the same value of the MCB channel prescaler. These prescalers must also be synchronized. In this case, A1 and B1 registers represent the same timing scale for duty cycle and dead-time insertion.

Figure 28-44 shows the load of A1 and B1 registers, which occurs when the selected counter bus reaches the value one. This counter value defines the cycle boundary. Values written to A2 or B2 within cycle (*n*) are loaded into A1 or B1 registers, respectively, and used to generate matches in cycle (*n* + 1).

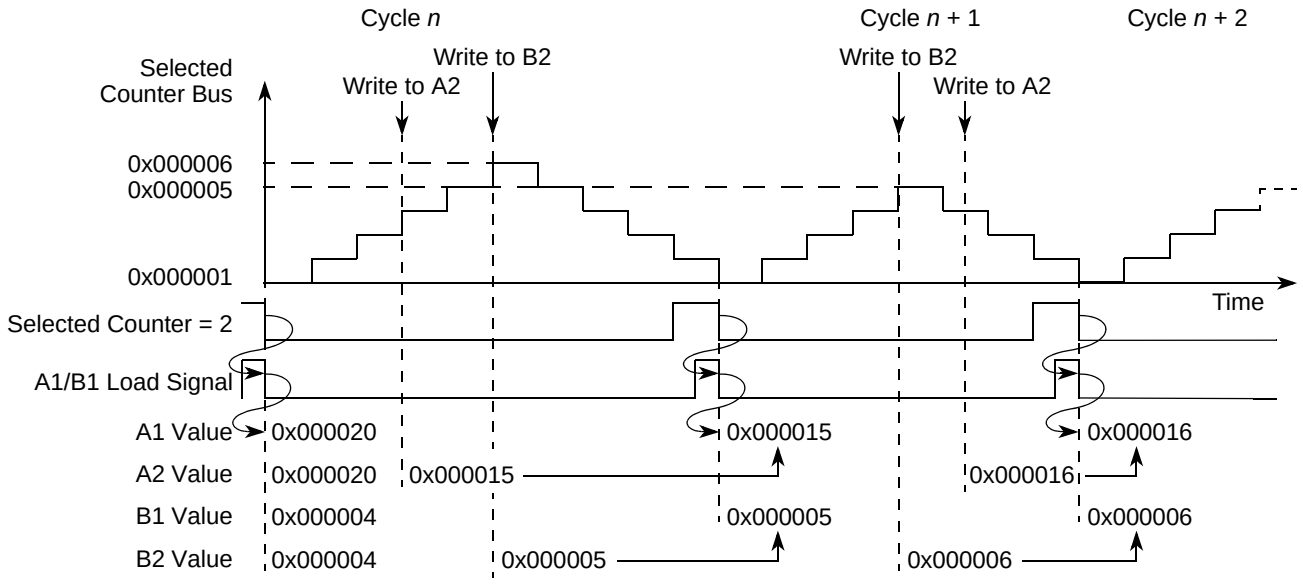


Figure 28-44. OPWMCB A1 and B1 Registers Load

The (OU[*n*] in EMIOS_OUDR) bit can be used to disable the A1 and B1 updates, thus allowing to synchronize the load on these registers with the load of A1 or B1 registers in others channels. Using the update disable bit, A1 and B1 registers can be updated at the same counter cycle, allowing both registers to change at the same time.

In this mode A1 matches always sets the internal counter to 0x00_0001. When operating with leading edge dead time insertion the first A1 match sets the internal counter to 0x00_0001. When a match occurs between register B1 and the internal time base, the output flip-flop is set to the value of the EDPOL bit. In the following match between register A1 and the selected time base, the output flip-flop is set to the complement of the EDPOL bit. This sequence repeats continuously. The internal counter should not reach 0x00_0000 as consequence of a rollover. To avoid this, the user must not write a value greater than twice the difference between external count up limit and EMIOS_CADR value to the EMIOS_CBDR register.

Figure 28-45 shows two cycles of a center-aligned PWM signal. Both A1 and B1 register values are changing within the same cycle, which allows to vary at the same time the duty cycle and dead-time values.

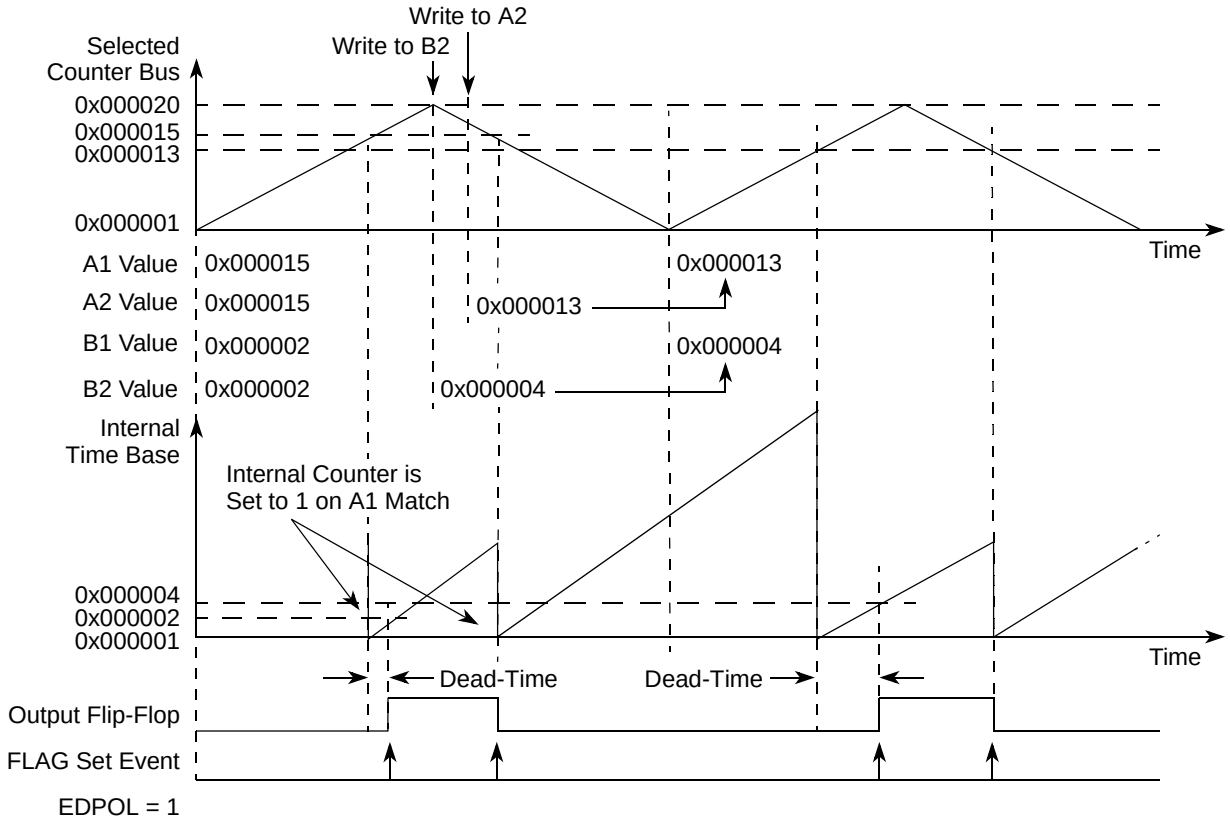


Figure 28-45. Output PWMCB with Lead Dead-Time Insertion

When operating with trailing edge dead-time insertion, the first match between A1 and the selected time base sets the output flip-flop to the value of the EDPOL bit and sets the internal counter to 0x00_0001. In the second match between register A1 and the selected time base, the internal counter is set to 0x00_0001 and B1 matches are enabled. When the match between register B1 and the selected time base occurs, the output flip-flop is set to the complement of the EDPOL bit. This sequence repeats continuously.

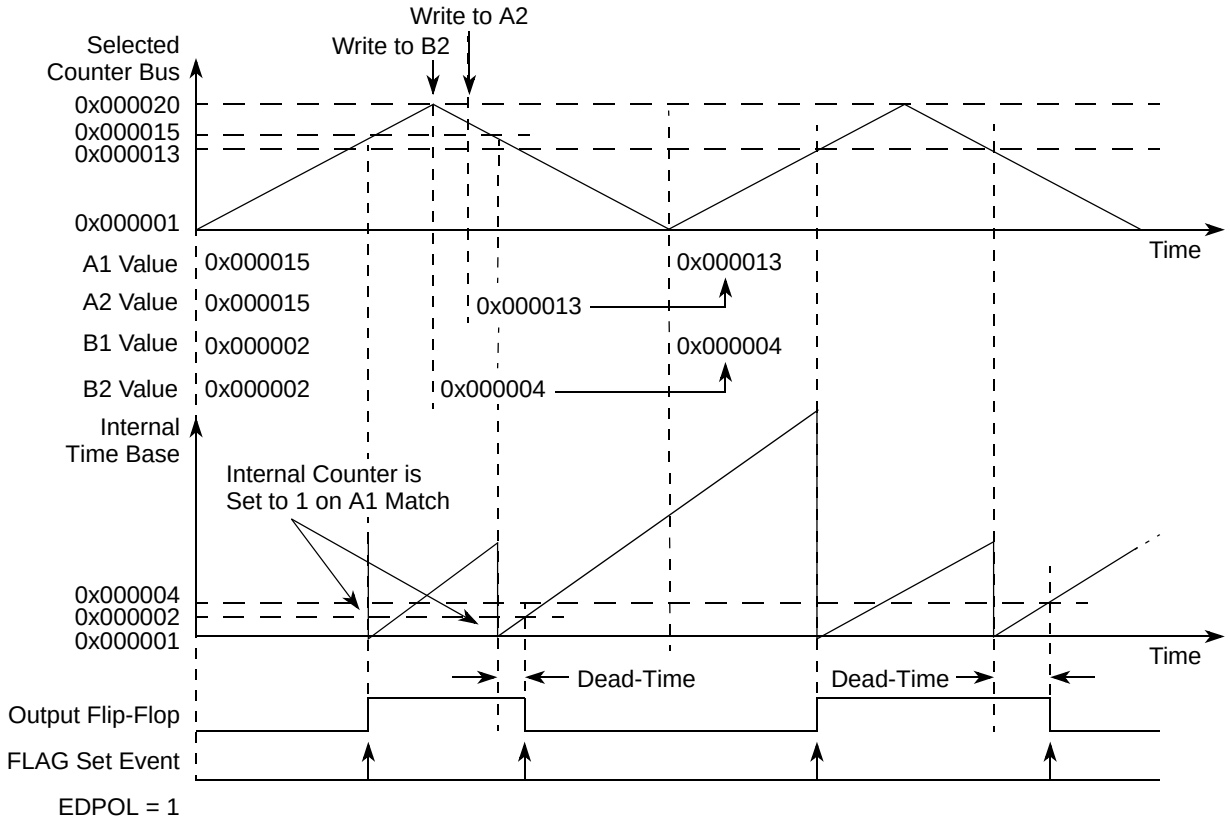


Figure 28-46. Output PWMCB with Trail Dead-Time Insertion

FLAG can be generated in the trailing edge of the output PWM signal when MODE[5] is cleared, or in both edges when MODE[5] is set. If subsequent matches occur on comparators A and B, the PWM pulses continue to be generated regardless of the state of the FLAG bit.

NOTE

In OPWMCB mode, FORCMA and FORCMB do not have the same behavior as a regular match. Instead they force the output flip-flop to constant value, which depends on the selected dead-time insertion mode, lead or trail and the value of the EDPOL bit.

FORCMA has different behaviors depending on the selected dead time insertion mode, lead or trail. In lead dead-time insertion FORCMA force a transition in the output flip-flop to the opposite of EDPOL. In trail dead-time insertion the output flip-flop is forced to the value of EDPOL bit.

If FORCMB bit is set, the output flip-flop value depends on the selected dead-time insertion mode. In lead dead time insertion FORCMB forces the output flip-flop to transition to EDPOL bit value. In trail dead-time insertion the output flip-flop is forced to the opposite of EDPOL bit value.

NOTE

FORCMA bit set does not set the internal time-base to 0x00_0001 as a regular A1 match.

The FLAG bit is not set either in case of a FORCMA or FORCMB or even if both forces are issued at the same time.

NOTE

FORCMA and FORCMB have the same behavior even in freeze or normal mode regarding the output pin transition.

When FORCMA is issued along with FORCMB, the output flip-flop is set to the opposite of EDPOL bit value. This is equivalent of saying that FORCMA has precedence over FORCMB when lead dead-time insertion is selected and FORCMB has precedence over FORCMA when trail dead-time insertion is selected.

Duty cycle from 0% to 100% can be generated by setting appropriate values to A1 and B1 registers relatively to the period of the external time base. Setting A1 = 1 generates a 100% duty cycle waveform. If A1 is greater than the maximum value of the selected counter bus period, then a 0% duty cycle is produced. Assuming EDPOL is set to one and OPWMCB mode with trail dead-time insertion, 100% duty cycle signals can be generated if B1 occurs at or after the cycle boundary (external counter = 1).

Only values different than 0x00_0000 are allowed to be written to A1 register. If 0x00_0000 is loaded to A1 the results are unpredictable.

NOTE

A special case occurs when A1 is set to (external counter bus period)/2, which is the maximum value of the external counter. In this case, the output flip-flop is constantly set to the EDPOL bit value.

The internal channel logic prevents matches from one cycle to propagate to the next cycle. In trail dead-time insertion B1 match from cycle (*n*) could eventually cross the cycle boundary and occur in cycle (*n* + 1). In this case B1 match is masked out and does not cause the output flip-flop to transition. Therefore matches in cycle (*n* + 1) are not affected by the late B1 matches from cycle (*n*).

Figure 28-47 shows a 100% duty cycle output signal generated by setting A1 = 4 and B1 = 3. In this case the trailing edge is positioned at the boundary of cycle *n* + 1, which is actually considered to belong to cycle *n* + 2 and therefore does not cause the output flip-flip to transition.

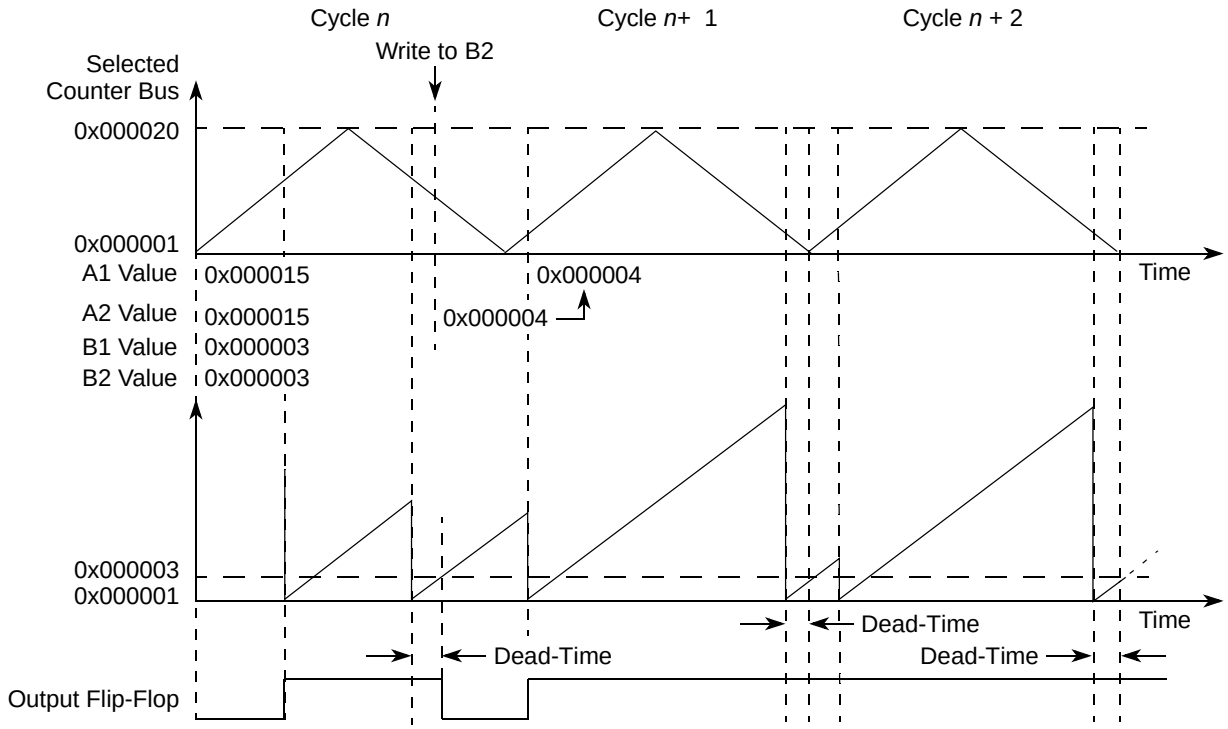


Figure 28-47. OPWMCB with 100% Duty Cycle (A1 = 4 and B1 = 3)

The output disable feature, if enabled, causes the output flip-flop to transition to the EDPOL inverted state. This feature allows an application to force the channel output pin to a “safe” state. The internal channel matches continue to occur even in this case, thus generating flags. As soon as the output disable is deasserted, the channel output pin is again controlled by the A1 and B1 matches. This process is synchronous, meaning that the output channel pin transitions on system clock edges only.

It is important to notice that, as in OPWMB and OPWFMB modes, the match signal used to set or clear the channel output flip-flop is generated on the deassertion of the channel combinational comparator output signal which compares the selected time base with A1 or B1 register values. Refer to Figure 28-39, which shows the delay from matches to output flip-flop transition in OPWFMB mode. The operation of OPWMCB mode is similar to OPWFMB regarding matches and output pin transition.

28.4.1.1.14 Pulse-Width Modulation Buffered (OPWMB) Mode

OPWMB mode is used to generate pulses with programmable leading- and trailing-edge placement. An external counter must be selected from one of the counter buses. The A1 register value defines the first edge and B1 defines the second edge. The output signal polarity is defined by the EDPOL bit. If EDPOL is 0, a negative edge occurs when A1 matches the selected counter bus; and a positive edge occurs when B1 matches the selected counter bus.

The A1 and B1 registers are double buffered and updated from A2 and B2, respectively, at the cycle boundary. The load operation is similar to the OPWFMB mode. Refer to Figure 28-41 for more information about A1 and B1 registers’ update.

FLAG can be generated at B1 matches, when MODE[5] is cleared, or on either A1 or B1 matches when MODE[5] is set. If subsequent matches occur on comparators A and B, the PWM pulses continue to be generated, regardless of the state of the FLAG bit.

FORCMA and FORCMB bits allow the software to force the output flip-flop to the level corresponding to a match on A1 or B1. The FLAG bit is not set by the FORCMA and FORCMB operations.

Some rules applicable to the OPWMB mode include:

- B1 matches have precedence over A1 matches if they occur at the same time within the same counter cycle
- A1 = 0 match from cycle(n) has precedence over B1 match from cycle(n – 1)
- A1 matches are masked out if they occur after B1 match within the same cycle
- Any value written to A2 or B2 on cycle(n) is loaded to A1 and B1 registers at the following cycle boundary (assuming (OU[n] in EMIOS_OUDR) is not asserted). The new values are used for A1 and B1 matches in cycle(n + 1)

Figure 28-48 shows the operation of the OPWMB mode regarding A1 and B1 matches and the transition of the channel output pin. In this example, EDPOL is set to 0.

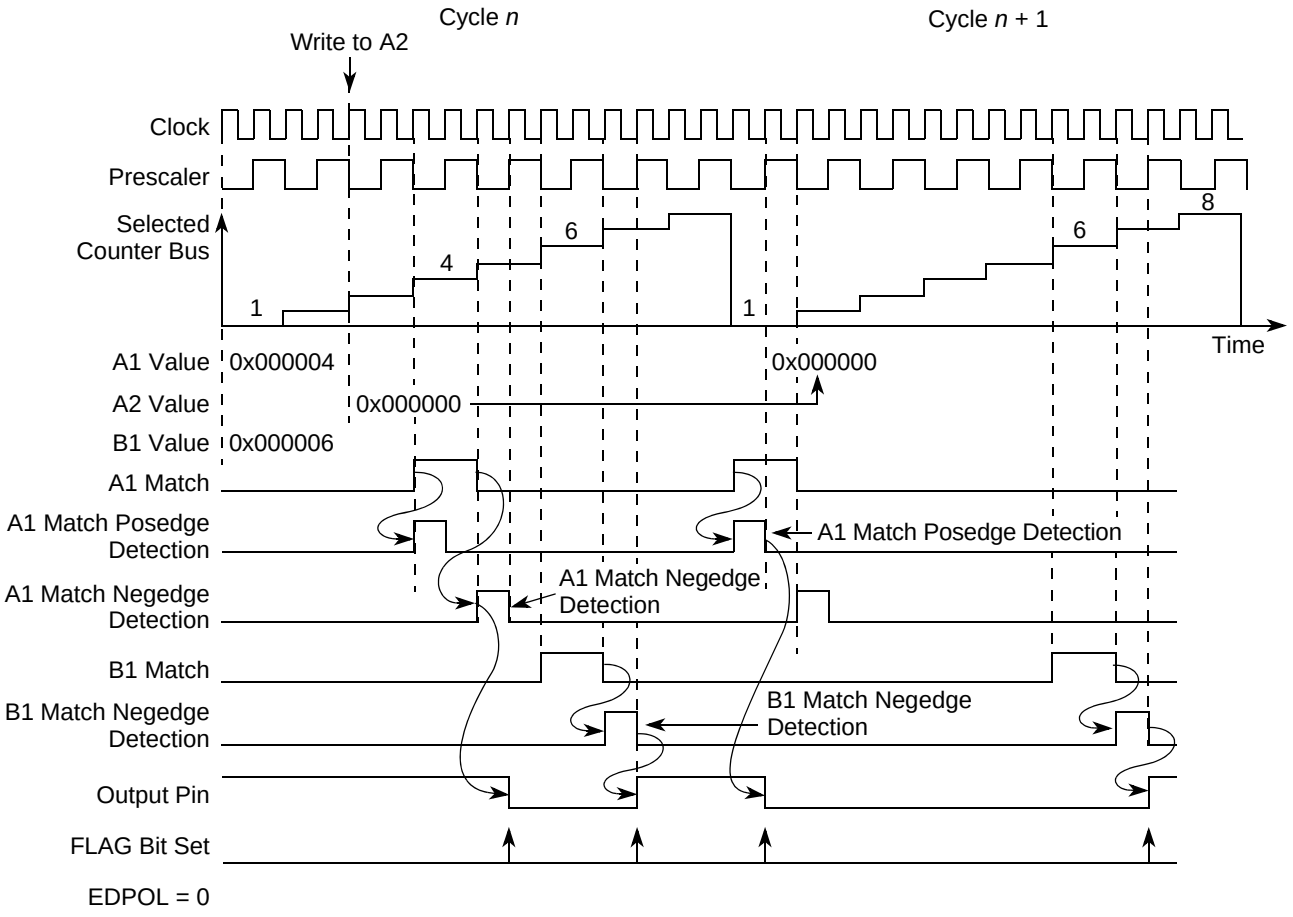


Figure 28-48. OPWMB Mode Matches and Flags

The output pin transitions are based on the negedges of the A1 and B1 match signals. Figure 28-48 shows in cycle(n + 1) the value of the A1 register being set to zero. In this case, the match posedge is used instead of the negedge to transition the output flip-flop.

Figure 28-49 shows the channel operation for 0% duty cycle. Note that the A1 match posedge signal occurs at the same time as the B1 = 8 negedge signal. In this case A1 match has precedence over B1 match, causing the output pin to remain at EDPOL bit value, thus generating a 0% duty cycle signal.

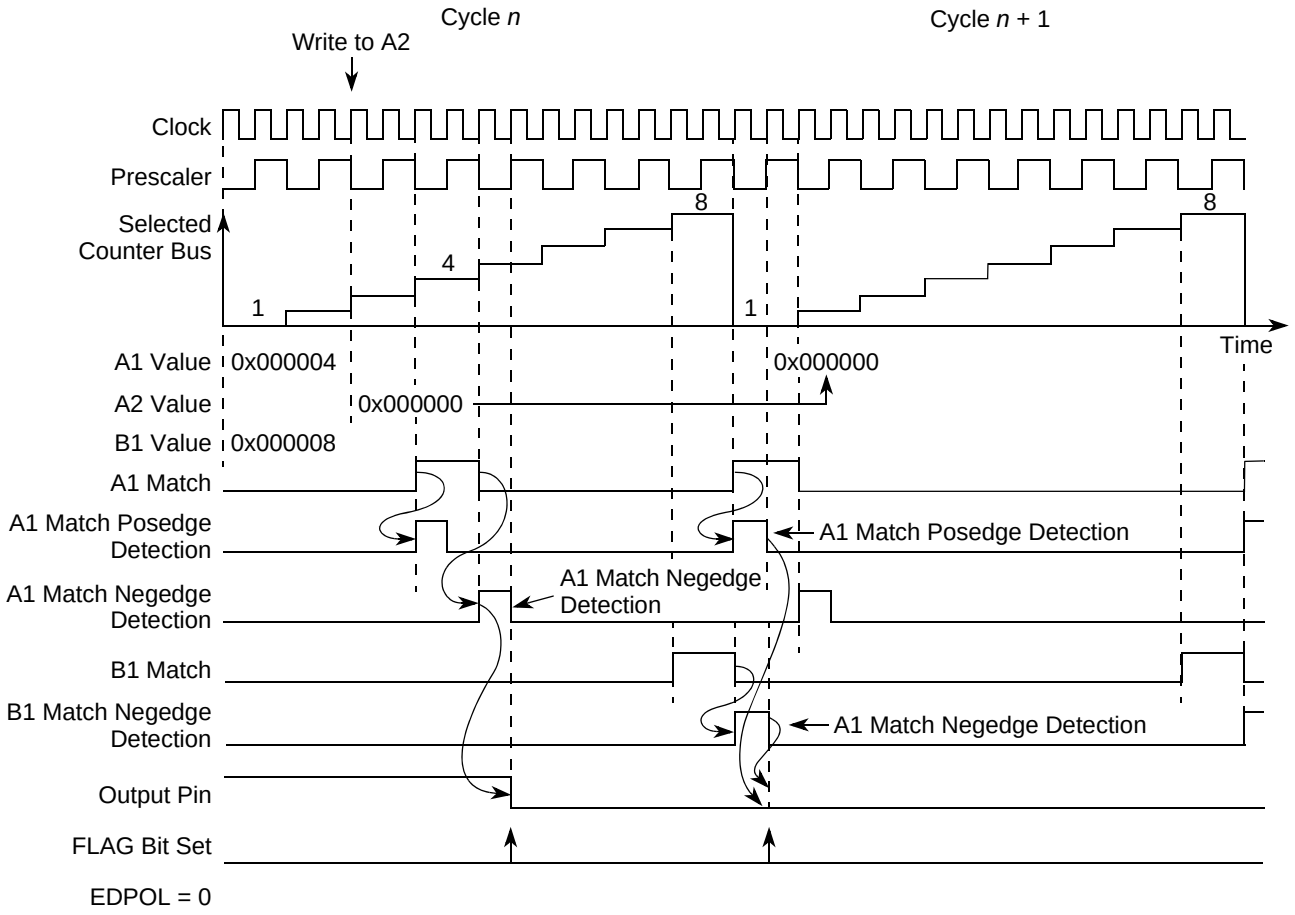


Figure 28-49. OPWMB Mode with 0% Duty Cycle

Figure 28-50 shows the operation of the OPWMB mode with the output disable signal asserted. The output disable forces a transition in the output pin to the EDPOL bit value. After deassertion, the output disable allows the output pin to transition at the following A1 or B1 match. The output disable does not modify the flag bit behavior. There is one system clock delay between the assertion of the output disable signal and the transition of the output pin to EDPOL.

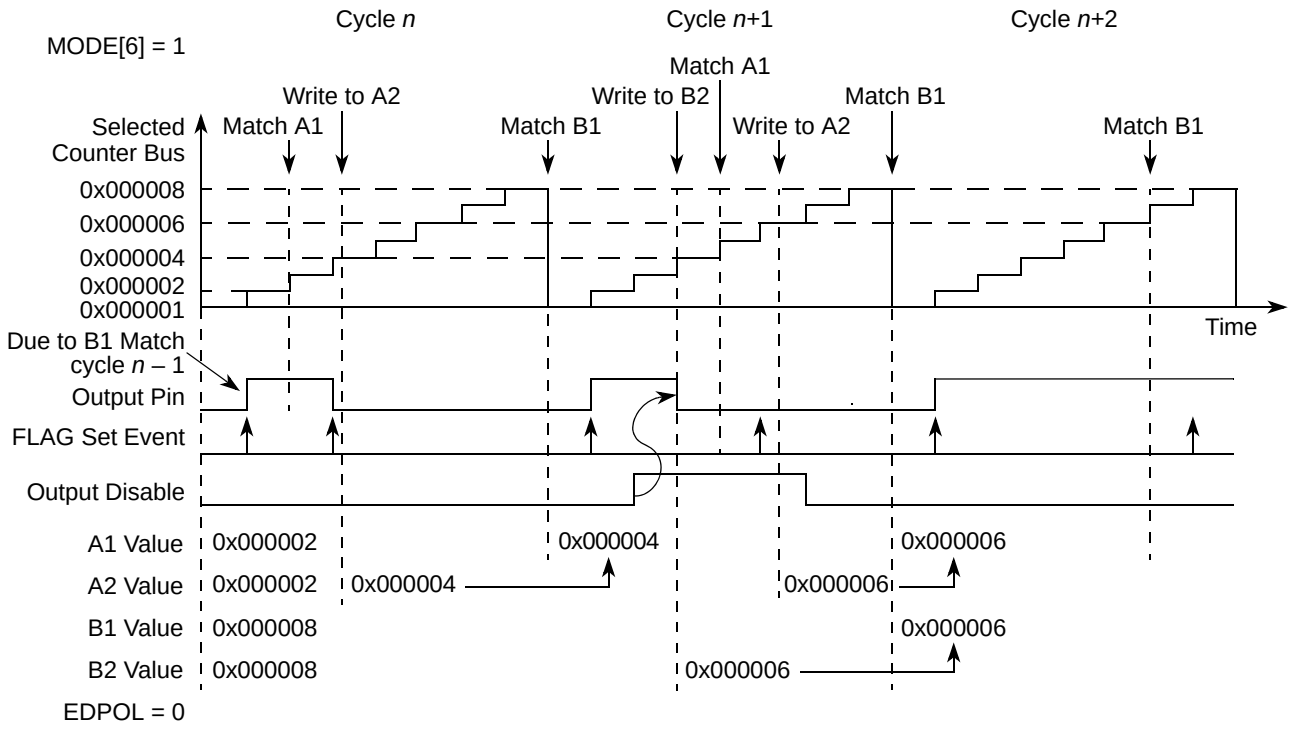


Figure 28-50. OPWMB Mode with Active Output Disable

Figure 28-51 shows a waveform changing from 100% to 0% duty cycle. In this case, EDPOL is 0. In this example, B1 is programmed to the same value as the period of the external selected time base.

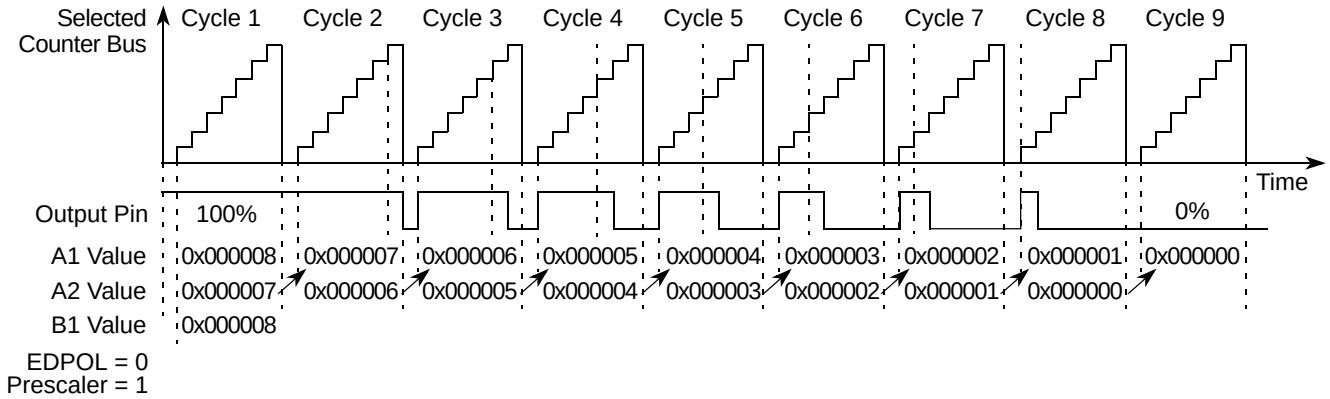


Figure 28-51. OPWMB Mode from 100% to 0% Duty Cycle

In Figure 28-51, if B1 is set to a value lower than 0x00_0008, it is not possible to achieve 0% duty cycle by changing only A1 register value. Because B1 matches have precedence over A1 matches the output pin transitions to the opposite of EDPOL bit at B1 match. If B1 is set to 0x00_0009, for instance, B1 match does not occur, thus a 0% duty cycle signal is generated.

28.4.1.1.15 Output Pulse Width Modulation with Trigger (OPWMT) Mode

OPWMT mode (MODE[0:6] = 010_0110) is intended to support the generation of Pulse Width Modulation signals where the period is not modified while the signal is being output, but where the duty

cycle is varied and must not create glitches. The mode is intended to be used in conjunction with other channels executing in the same mode and sharing a common timebase. It supports each channel with a fixed PWM leading edge position with respect to the other channels and the ability to generate a trigger signal at any point in the period that can be output from the module to initiate activity in other parts of the device, such as starting ADC conversions.

An external counter driven in either MC Up or MCB Up mode must be selected from one of the counter buses.

Register A1 defines the leading edge of the PWM output pulse and as such the beginning of the PWM's period. This makes it possible to ensure that the leading edge of multiple channels in OPWMT mode can occur at a specific time with respect to the other channels when using a shared timebase. This can allow the introduction of a fixed offset for each channel which can be particularly useful in the generation of lighting PWM control signals where it is desirable that edges are not coincident with each other to help eliminate noise generation. The value of register A1 represents the shift of the PWM channel with respect to the selected timebase. A1 can be configured with any value within the range of the selected time base. Note that registers loaded with 0x00_0000 do not produce matches if the timebase is driven by a channel in MCB mode.

A1 is not buffered as the shift of a PWM channel must not be modified while the PWM signal is being generated. In case A1 is modified it is immediately updated and one PWM pulse could be lost.

EMIOS_CBDR[*n*] address gives access to B2 register for write and B1 register for read. Register B1 defines the trailing edge of the PWM output pulse and as such the duty cycle of the PWM signal. To synchronize B1 update with the PWM signal and so ensure a correct output pulse generation the transfer from B2 to B1 is done at every match of register A1. This behavior is the same as the OPWM mode with next period update.

EMIOS_OUDR register affects transfers between B2 and B1 only.

In order to account for the shift in the leading edge of the waveform defined by register A1, it is necessary that the trailing edge, held in register B1, can roll over into the next period. This means that a match against the B1 register should not have to be qualified by a match in the A1 register. The impact of this would mean that incorrectly setting register B1 to a value less that register A1 results in the output being held over a cycle boundary until the B1 value is encountered.

This mode provides a buffered update of the trailing edge by updating register B1 with register B2 contents only at a match of register A1.

The value loaded in register A1 is compared with the value on the selected time base. When a match on comparator A1 occurs, the output flip-flop is set to the value of the EDPOL bit. When a match occurs on comparator B, the output flip-flop is set to the complement of the EDPOL bit.

Note that the output pin and flag transitions are based on the posedges of the A1, B1 and A2 match signals. Please, refer to Figure 28-48 at Section 28.4.1.1.15, Output Pulse Width Modulation with Trigger (OPWMT) Mode for details on match posedge.

Register A2 defines the generation of a trigger event within the PWM period and A2 should be configured with any value within the range of the selected time base, otherwise no trigger is generated. A match on the comparator generates the FLAG signal but it has no effect on the PWM output signal generation. The

typical setup to obtain a trigger with FLAG is enabling DMA and driving the channel's ipd_done input high.

A2 is not buffered and therefore its update is immediate. If the channel is running when a change is made this could cause either the loss of one trigger event or the generation of two trigger events within the same period. Register A2 can be accessed by reading or writing the eMIOS200 UC Alternate A Register (EMIOS_ALTA) at UC[*n*] base address + 0x0014.

FLAG signal is set only at match on the comparator with A2. A match on the comparator with A1 or B1 or B2 has no effect on FLAG.

At any time, the FORCMA and FORCMB bits allow the software to force the output flip-flop to the level corresponding to a match on A or B respectively. Any FORCMA and/or FORCMB has priority over any simultaneous match regarding to output pin transitions. Note that the load of B2 content on B1 register at an A match is not inhibited due to a simultaneous FORCMA/FORCMB assertion. If both FORCMA and FORCMB are asserted simultaneously the output pin goes to the opposite of EDPOL value such as if A1 and B1 registers had the same value. FORCMA assertion causes the transfer from register B2 to B1 such as a regular A match, regardless of FORCMB assertion.

If subsequent matches occur on comparators A1 and B, the PWM pulses continue to be generated, regardless of the state of the FLAG bit.

At OPWMT mode entry the output flip-flop is set to the complement of the EDPOL bit in the EMIOS_CCR[*n*] register.

In order to achieve 0% duty cycle both registers A1 and B must be set to the same value. When a simultaneous match on comparators A and B occur, the output flip-flop is set at every period to the complement value of EDPOL.

In order to achieve 100% duty cycle the register B1 must be set to a value greater than maximum value of the selected time base. As a consequence if 100% duty cycle must be implemented the maximum counter value for the time base is 0xFF_FFFE for a 24-bit counter and respectively 0xFFFE for a 16-bit counter. When a match on comparator A1 occurs the output flip-flop is set at every period to the value of EDPOL bit. The transfer from register B2 to B1 is still triggered by the match at comparator A.

As with other eMIOS200 mode, the OPWMT mode implements the Output Disable function. Setting the ODIS bit in EMIOS_CCR[*n*] enables the Output Disable function. If the selected Output Disable input signal is asserted for the channel, the output pin goes to the inverse of the EDPOL. The channel continues to operate normally, although the output is fixed. When the Output Disable input signal is negated, the output pin returns to operate as normal.

Figure 28-52 shows the Unified Channel running in OPWMT mode with Trigger Event Generation and duty cycle update on next period update.

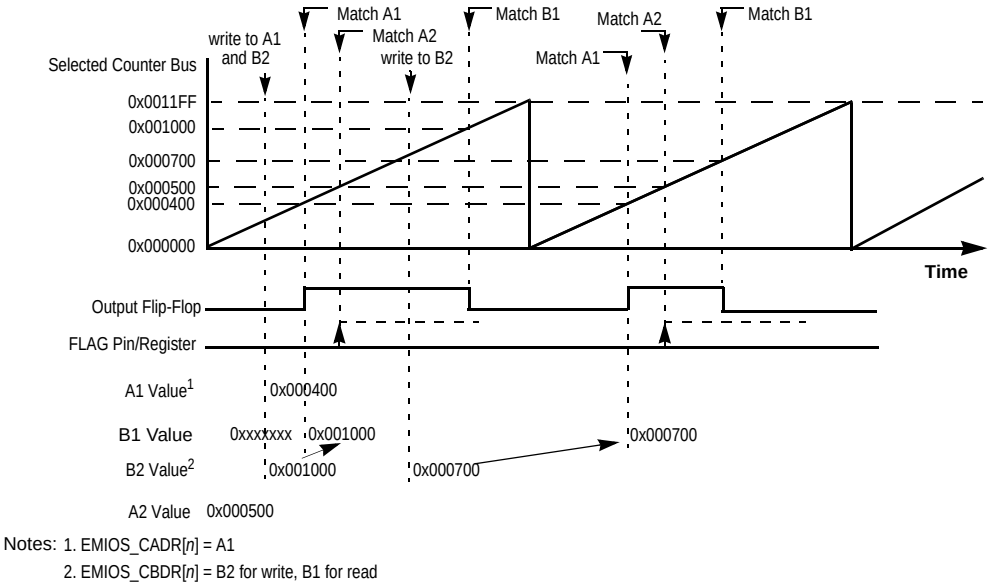


Figure 28-52. OPWMT Example{statement}

Figure 28-53 shows the Unified Channel running in OPWMT mode with Trigger Event Generation and 0% duty cycle.

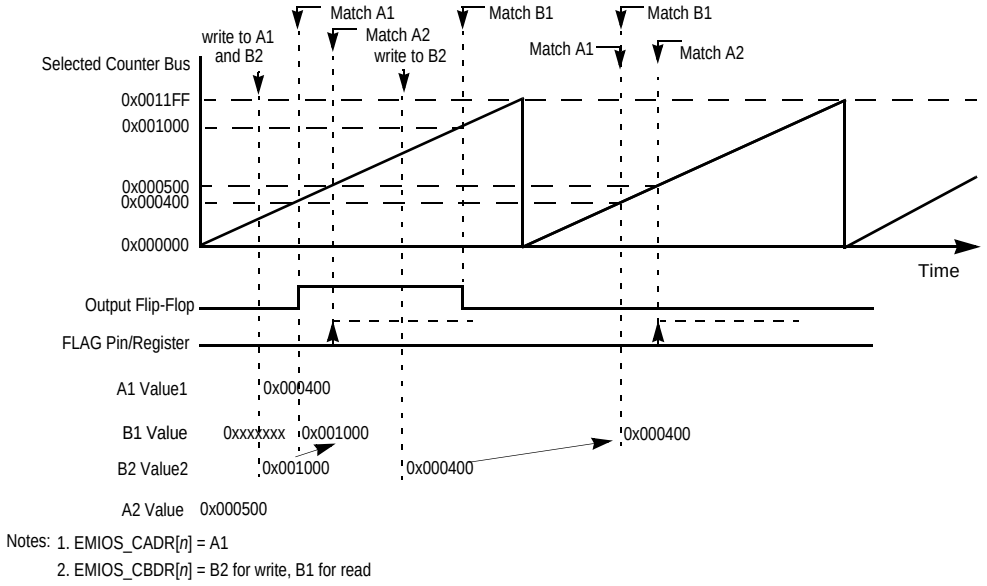


Figure 28-53. OPWMT with 0% Duty Cycle

Figure 28-54 shows the Unified Channel running in OPWMT mode with Trigger Event Generation and 100% duty cycle.

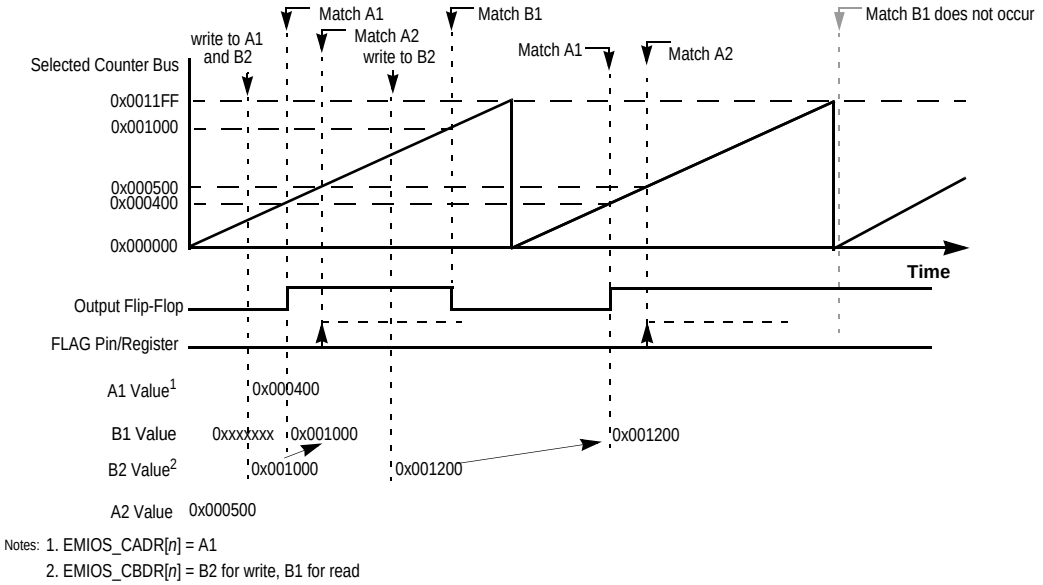


Figure 28-54. OPWMT with 100% Duty Cycle

28.4.1.2 Input Programmable Filter (IPF)

The IPF ensures that only valid input pin transitions are received by the unified channel edge detector. A block diagram of the IPF is shown in Figure 28-55.

The IPF is a 5-bit programmable up counter that is incremented by the selected clock source, according to IF bits in EMIOS_CCR[*n*].

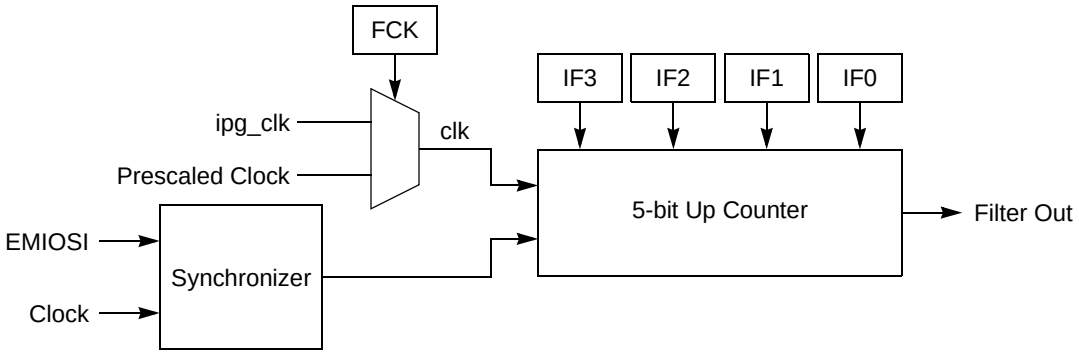


Figure 28-55. Input Programmable Filter Submodule Diagram

The input signal is synchronized by system clock. When a state change occurs in this signal, the 5-bit counter starts counting up. As long as the new state is stable on the pin, the counter remains incrementing. If a counter overflow occurs, the new pin value is validated. In this case, it is transmitted as a pulse edge to the edge detector. If the opposite edge appears on the pin before validation (overflow), the counter is reset. At the next pin transition, the counter starts counting again. Any pulse that is shorter than a full range of the masked counter is regarded as a glitch and it is not passed on to the edge detector. Figure 28-56 shows a timing diagram of the input filter.

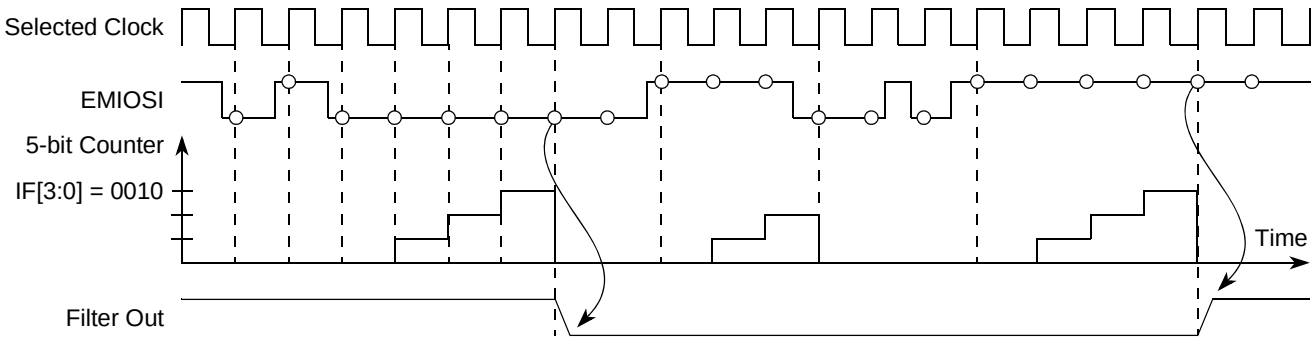


Figure 28-56. Input Programmable Filter Example

28.4.1.3 Clock Prescaler (CP)

The CP divides the GCP output signal to generate a clock enable for the internal counter of the unified channels. It is a programmable 2-bit down counter. The GCP output signal is prescaled by the value defined in the UCPRE bits in the EMIOS_CCR[*n*] register. The output is clocked every time the counter reaches zero. Counting is enabled by setting the UCPREN bit in the EMIOS_CCR[*n*]. The counter can be stopped at any time by clearing this bit, thereby stopping the internal counter in the unified channel.

In order to ensure safe working and avoid glitches the following steps must be performed whenever any update in the prescaling rate is desired:

1. Write 0 at both GPREN bit in EMIOSMCR register and UCPREN bit in EMIOS_CCR[*n*] register, thus disabling prescalers;
2. Write the desired value for prescaling rate at UCPRE[] bits in EMIOS_CCR[*n*] register;
3. Enable channel prescaler by writing 1 at UCPREN bit in EMIOS_CCR[*n*] register;
4. Enable global prescaler by writing 1 at GPREN bit in EMIOSMCR register.

The prescaler is not disabled during either freeze state or negated GTBE input.

28.4.1.4 Effect of Freeze on the Unified Channel

When in debug mode, if the FRZ bit in the EMIOS_MCR register and the FREN bit in the EMIOS_CCR[*n*] are both set, the internal counter and unified channel capture and compare functions are halted. The unified channel is frozen in its current state.

During freeze, all registers are accessible. When the unified channel is operating in an output mode, the force match functions remain available, allowing the software to force the output to the desired level.

During input modes, any input events that may occur while the channel is frozen are ignored.

When exiting debug mode or when the freeze enable bit is cleared (FRZ in the EMIOS_MCR or FREN in the EMIOS_CCR[*n*] register), the channel actions resume but may be inconsistent until the channel enters GPIO mode again.

28.4.2 IP Bus Interface Unit (BIU)

The BIU provides the interface between the internal interface bus (IIB) and the peripheral bus, allowing communication among all submodules and this IP interface.

The BIU allows 8-, 16-, and 32-bit access. They are performed over a 32-bit data bus in a single cycle clock.

28.4.2.1 Effect of Freeze on the BIU

When the FRZ bit in the EMIOS_MCR register is set and the module is in debug mode, the operation of BIU is not affected.

28.4.3 Global Clock Prescaler Submodule (GCP)

The GCP divides the system clock to generate a clock for the CPs of the unified channels. It is a programmable 8-bit up counter. The main clock signal is prescaled by the value defined in the GPRE bits in EMIOS_MCR. The output is clocked every time the counter overflows. Counting is enabled by setting the GPREN bit in the EMIOS_MCR. The counter can be stopped at any time by clearing this bit, thereby stopping the internal counter in all the unified channels.

28.4.3.1 Effect of Freeze on the GCP

When the FRZ bit in the EMIOS_MCR register is set and the module is in debug mode, the operation of GCP submodule is not affected, i.e., there is no freeze function in this submodule.

28.5 Reset

The eMIOS200 is reset by the global asynchronous system reset signal.

The MDIS bit in the EMIOS_MCR register and the UCDIS bits in the EMIOS_UCDIS registers are cleared during reset.

On resetting the eMIOS200 all unified channels enter GPIO input mode.

28.6 Interrupts

The eMIOS200 can generate one interrupt per channel. An interrupt request is generated according to the configuration of the channel and input events or matches. See Chapter 10, Interrupts and Interrupt Controller (INTC), for details on the eMIOS200 interrupt vector.

28.7 DMA Requests

The connection of the eMIOS200 DMA request signals to the DMA channel mux is described in Section 23.5.2, Enabling and Configuring Sources.

28.8 Initialization/Application Information

On resetting the eMIOS200 all unified channels enter GPIO input mode.

28.8.1 Considerations

Before changing an operating mode, the unified channel must be programmed to GPIO mode and EMIOS_CADR[*n*] and EMIOS_CBDR[*n*] registers must be updated with the correct values for the next operating mode. Then the EMIOS_CCR[*n*] register can be written with the new operating mode. If a unified channel is changed from one mode to another without performing this procedure, the first operation cycle of the selected time base can be random, i.e., matches can occur in random time if the contents of EMIOS_CADR[*n*] or EMIOS_CBDR[*n*] were not updated with the correct value before the time base matches the previous contents of EMIOS_CADR[*n*] or EMIOS_CBDR[*n*].

When interrupts are enabled, the software must clear the FLAG bits before exiting the interrupt service routine.

28.8.2 Application Information

Correlated output signals can be generated by all output operation modes. The OU[*n*] bits in EMIOS_OUDR can be used to control the update of these output signals.

In order to guarantee the internal counters of correlated channels are incremented in the same clock cycle, the internal prescalers must be set before enabling the global prescaler. If the internal prescalers are set after enabling the global prescaler, the internal counters may increment in the same ratio but at a different clock cycle.

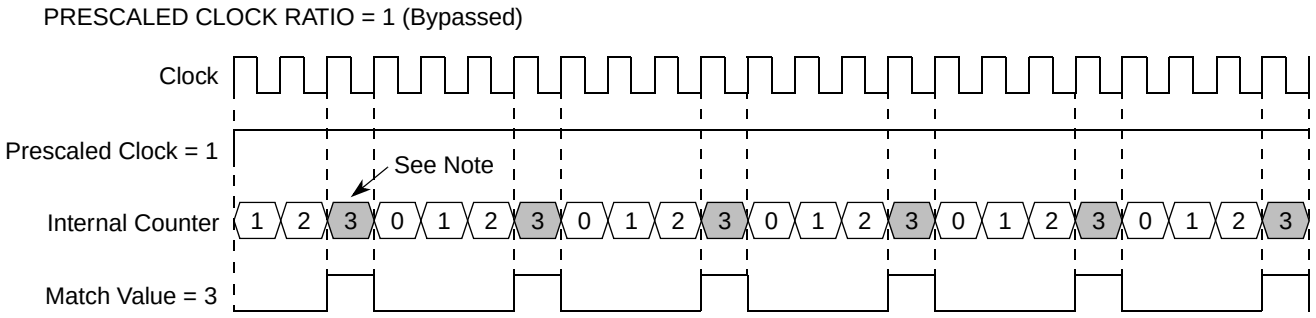
It is recommended to drive output disable input signals with the emios_flag_out signals of some unified channels running in SAIC mode. When an output disable condition happens, the software interrupt routine must service the output channels before servicing the channels running SAIC. This procedure avoids glitches in the output pins.

28.8.3 Time Base Generation

For MC, OPWFM, and OPWM with internal clock source operation modes, the internal counter rate can be modified by configuring the clock prescaler ratio. Figure 28-57 shows an example of a time base with prescaler ratio equal to one. When the prescaler is greater than one, the counter is immediately cleared on a match and then incremented in the next prescaled clock edge, except when running in OPWFM mode or MC mode with internal clock source. In these cases, the counter will skip the next prescaled clock edge and continue incremented on subsequent edges, as shown in Figure 28-58.

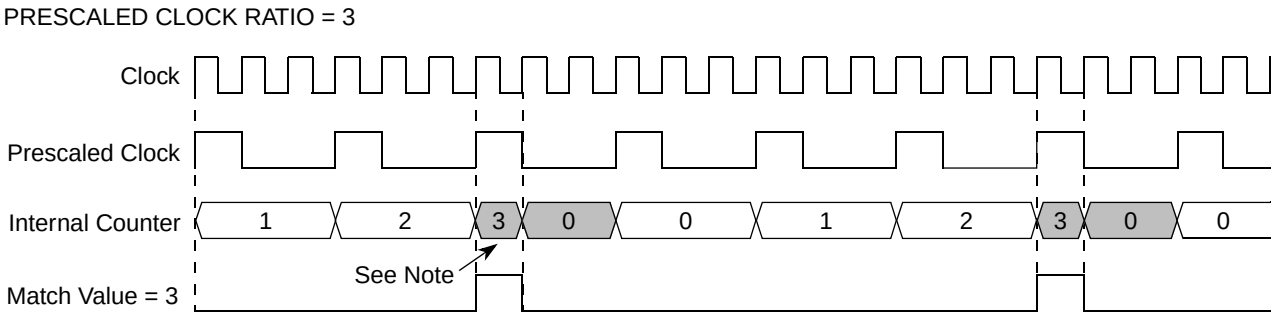
NOTE

MCB, OPWFMB, and OPWMB modes have a different behavior.



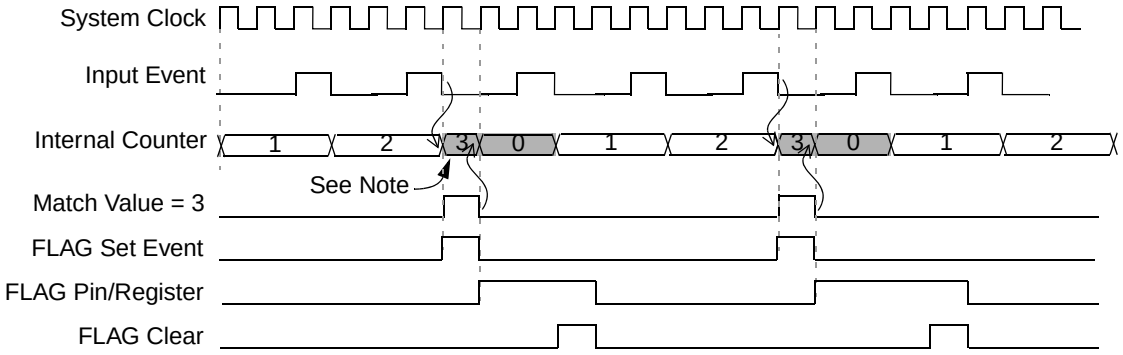
Note: When a match occurs, the first clock cycle is used to clear the internal counter, starting another period.

Figure 28-57. Time Base Period when Running in the Fastest Prescaler Ratio



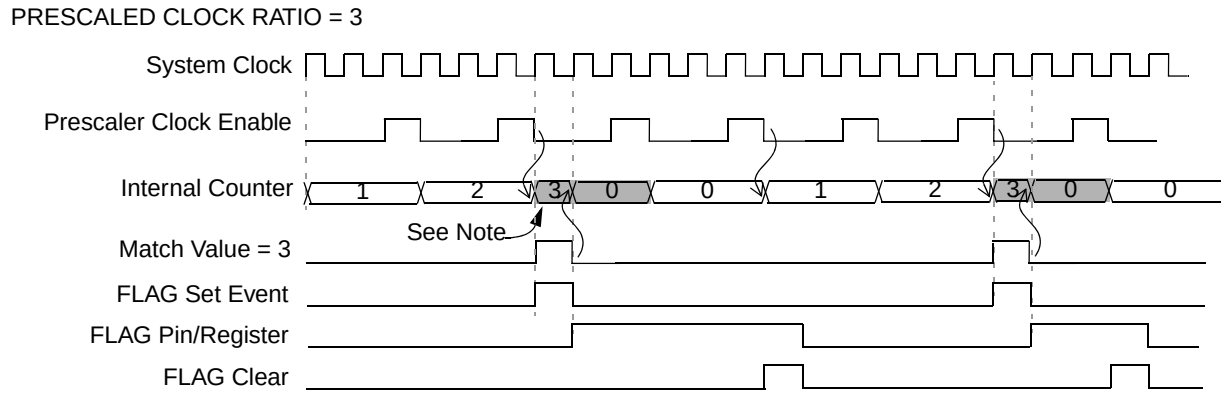
Note: When a match occurs, the first clock cycle is used to clear the internal counter, and only after a second edge of prescaled clock the counter will start counting.

Figure 28-58. Time Base Period when Running with a Prescaler Ratio Greater Than 1



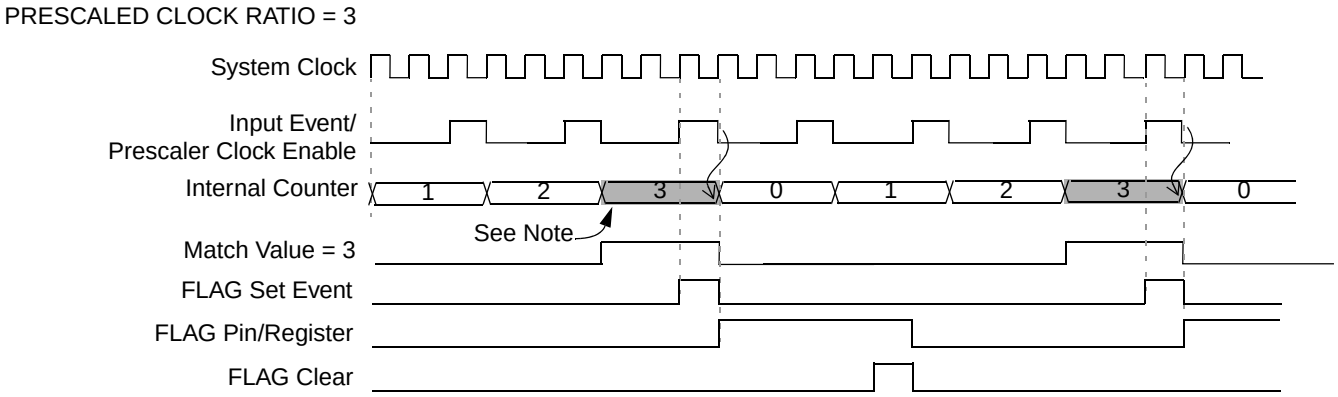
Note: When a match occurs, the first system clock cycle is used to clear the internal counter, and at the next edge of prescaler clock enable the counter will start counting.

Figure 28-59. Time Base Generation with External Clock and Clear on Match Start



Note: When a match occurs, the first clock cycle is used to clear the internal counter, and only after a second edge of prescaled clock the counter will start counting.

Figure 28-60. Time Base Generation with Internal Clock and Clear on Match Start



Note: The match occurs only when the input event/prescaler clock enable is active. Then, the internal counter is immediately cleared.

Figure 28-61. Time Base Generation with Clear on Match End

28.8.4 Coherent Accesses

For IPWM and IPM modes, it is recommended that the software wait for a new FLAG set event before reading EMIOS_CADR[*n*] and EMIOS_CBDR[*n*] registers to get a new measurement. The FLAG indicates that new data has been captured and it is the only way to assure data coherency.

The FLAG set event can be detected by polling the FLAG bit or by enabling the interrupt or DMA request generation.

Reading the EMIOS_CADR[*n*] register again in the same period of the last read of EMIOS_CBDR[*n*] register may lead to incoherent results. This occurs if the last read of EMIOS_CBDR[*n*] register occurred after a disabled B2 to B1 transfer.

Chapter 29
Controller Area Network (FlexCAN)

29.1 Introduction

The PXN20 contains as many as six controller area network (FlexCAN) blocks. Each FlexCAN module is a communication controller implementing the CAN protocol according to Bosch Specification version 2.0B and ISO Standard 11898. The CAN protocol is used as a serial data bus, meeting the specific requirements of this field: real-time processing, reliable operation in the EMI environment of a vehicle, cost-effectiveness and required bandwidth.

NOTE

The FlexCAN_F block is not implemented on the PXN21.

The CAN protocol interface (CPI) sub-module manages the serial communication on the CAN bus, requesting RAM access for receiving and transmitting message frames, validating received messages and performing error handling. The message buffer management (MBM) sub-module handles message buffer selection for reception and transmission, taking care of arbitration and ID matching algorithms. The bus interface unit (BIU) sub-module controls the access to and from the internal interface bus, to establish connection to the CPU and other blocks. Clocks, address and data buses, interrupt outputs, and test signals are accessed through the bus interface unit.

29.1.1 Block Diagram

A simplified block diagram of the FlexCAN illustrates the functionality and interdependence of major sub-blocks (see Figure 29-1).

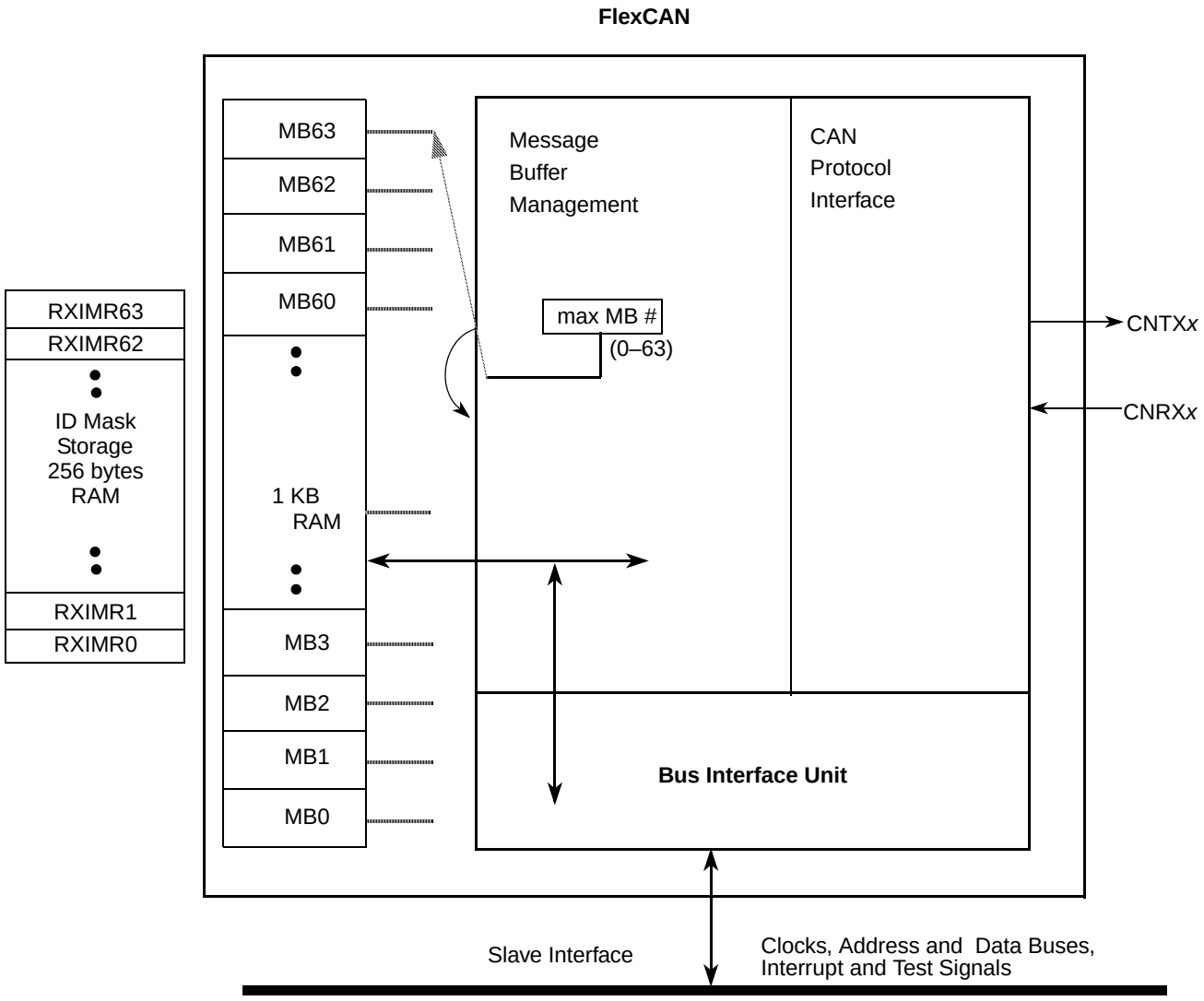


Figure 29-1. FlexCAN Block Diagram

29.1.2 Features

The FlexCAN has these major features:

- Full implementation of the CAN protocol specification, Version 2.0A/B
  - — Standard data and remote frames
  - — Extended data and remote frames
  - — Zero to eight bytes data length
  - — Programmable bit rate as high as 1 Mbit/s
  - — Content-related addressing
- 64 flexible message buffers (MBs) of 0 to 8 bytes data length
- Each message buffer configurable as Rx or Tx, all supporting standard and extended messages

- Individual Rx mask registers per message buffer
- Includes 1056 bytes of RAM used for message buffer storage
- Includes 256 bytes of RAM used for individual Rx mask registers
- Full featured Rx FIFO with storage capacity for six frames and internal pointer handling
- Powerful Rx FIFO ID filtering, capable of matching incoming IDs against either eight extended, 16 standard, or 32 partial (8 bits) IDs, with individual masking capability
- Selectable backwards compatibility with previous FlexCAN version
- Programmable clock source to the CAN protocol interface, either bus clock or crystal oscillator
- Unused message buffer and Rx mask register space can be used as general-purpose RAM space
- Listen-only mode capability
- Programmable loop-back mode supporting self-test operation
- Programmable transmission priority scheme: lowest ID, lowest buffer number or local priority on individual Tx message buffers.
- Hardware cancellation on Tx message buffers.
- Time stamp based on 16-bit free-running timer
- Global network time, synchronized by a specific message
- Maskable interrupts
- Independent of the transmission medium (an external transceiver is assumed)
- Short latency time due to an arbitration scheme for high-priority messages
- Low-power modes

29.1.3 Modes of Operation

There are four main operating modes of FlexCAN: normal, freeze, listen-only, and loop-back. One low-power mode is supported: module disable. For more details, refer to Section 29.4.8, Modes of Operation Details.

29.1.3.1 Normal Mode

In normal mode the module operates receiving and/or transmitting message frames, errors are handled normally and all the CAN protocol functions are enabled. In the MCU, there is no distinction between user and supervisor modes.

29.1.3.2 Freeze Mode

Freeze mode is entered when the FRZ bit in the module configuration register (CAN*x*_MCR) is asserted, while the HALT bit in CAN*x*_MCR is set, or if debug mode is requested by either core. In freeze mode no transmission or reception of frames is done, and synchronicity to the CAN bus is lost.

29.1.3.3 Listen-Only Mode

In this mode, transmission is disabled, all error counters are frozen, and the module operates in a CAN error passive mode. Only messages acknowledged by another CAN station are received. If FlexCAN detects a message that has not been acknowledged, it flags a BIT0 error (without changing the REC), as if it was trying to acknowledge the message.

29.1.3.4 Loop-Back Mode

The module enters this mode when the LPB bit in the control register is asserted. In this mode, FlexCAN performs an internal loop back that can be used for self test operation. The bit stream output of the transmitter is internally fed back to the receiver input. The Rx CAN input pin is ignored and the Tx CAN output goes to the recessive state (logic 1). FlexCAN behaves as it normally does when transmitting, and treats its own transmitted message as a message received from a remote node. In this mode, FlexCAN ignores the bit sent during the ACK slot in the CAN frame acknowledge field to ensure proper reception of its own message. Transmit and receive interrupts are generated.

29.1.3.5 Module-Disabled Mode

This low-power mode is entered when the MDIS bit in the CAN*x*_MCR register is asserted. When disabled, the clocks to the CAN protocol interface and message buffer management sub-modules are shut down. Exit from this mode is done by negating the MDIS bit in the CAN*x*_MCR register.

29.1.3.6 Halt Mode

This low power mode is entered when the corresponding HLT bit in the SIU_HLT0 is asserted. The HLT bit drives the stop input to the module. When in halt mode, the module puts itself in an inactive state and then informs the CPU that its clock can be shut down.

Exit from this mode happens when the HLT bit is de-asserted.

29.2 External Signal Description

Please refer to Table 3-1 and Section 3.4, Detailed Signal Description, for a complete description of the FlexCAN signals.

29.3 Memory Map and Registers

This section provides a detailed description of all FlexCAN registers.

29.3.1 Module Memory Map

The complete memory map for an individual FlexCAN module is shown in Table 29-1. Except for the base addresses, all FlexCAN modules have identical memory maps.

The Rx global mask (CAN*x*_RXGMASK), Rx buffer 14 mask (CAN*x*_RX14MASK) and the Rx buffer 15 mask (CAN*x*_RX15MASK) registers are provided for backwards compatibility, and are not used when the BCC bit in CAN*x*_MCR is asserted.

The offset address ranges 0x0060–0x047F and 0x0880–0x097F are occupied by two separate embedded memories. These two ranges are completely occupied by RAM (1 KB and 256 bytes, respectively) when FlexCAN is configured with 64 MBs. Furthermore, if the BCC bit in CAN*x*_MCR is negated, then the whole Rx individual mask registers address range (0x0880–0x097F) is considered reserved space.

Table 29-1. FlexCAN Memory Map

Offset from FlexCAN_BASE (FlexCAN_A: 0xFFFC_0000 FlexCAN_B: 0xFFFC_4000 FlexCAN_C: 0xFFFC_8000 FlexCAN_D: 0xFFFC_C000 FlexCAN_E: 0xFFFD_0000 FlexCAN_F: 0xFFFD_4000)	Register	Access	Reset Value	Section/Page
0x0000	CAN*x*_MCR—Module Configuration	R/W	0x9890_000F	29.3.4.1/29-11
0x0004	CAN*x*_CTRL—Control Register	R/W	0x0000_0000	29.3.4.2/29-14
0x0008	CAN*x*_TIMER—Free-Running Timer	R/W	0x0000_0000	29.3.4.3/29-17
0x000C	Reserved			
0x0010	CAN*x*_RXGMASK—Rx Global Mask	R/W	0xFFFF_FFFF	29.3.4.4.1/29-18
0x0014	CAN*x*_RX14MASK—Rx Buffer 14 Mask	R/W	0xFFFF_FFFF	29.3.4.4.2/29-19
0x0018	CAN*x*_RX15MASK—Rx Buffer 15 Mask	R/W	0xFFFF_FFFF	29.3.4.4.3/29-19
0x001C	CAN*x*_ECR—Error Counter Register	R/W	0x0000_0000	29.3.4.5/29-20
0x0020	CAN*x*_ESR—Error and Status Register	R/W	0x0000_0000	29.3.4.6/29-21
0x0024	CAN*x*_IMASK2—Interrupt Masks 2	R/W	0x0000_0000	29.3.4.7/29-23
0x0028	CAN*x*_IMASK1—Interrupt Masks 1	R/W	0x0000_0000	29.3.4.8/29-24
0x002C	CAN*x*_IFLAG2—Interrupt Flags 2	R/W	0x0000_0000	29.3.4.9/29-24
0x0030	CAN*x*_IFLAG1—Interrupt Flags 1	R/W	0x0000_0000	29.3.4.10/29-25
0x0034–0x007F	Reserved			
0x0080–0x017F	MB0–MB15—Message Buffers	R/W	—[1]	29.3.2/29-7
0x0180–0x027F	MB16–MB31—Message Buffers	R/W	—[1]	
0x0280–0x047F	MB32–MB63—Message Buffers	R/W	—[1]	
0x0480–0x087F	Reserved			
0x0880–0x08BF	CAN*x*_RXIMR0–CAN*x*_RXIMR15—Rx Individual Mask Registers	R/W	0x0000_0000	29.3.4.11/29-26
0x08C0–0x08FF	CAN*x*_RXIMR16–CAN*x*_RXIMR31—Rx Individual Mask Registers	R/W	0x0000_0000	
0x0900–0x097F	CANx_RXIMR32–CAN*x*_RXIMR63—Rx Individual Mask Registers	R/W	0x0000_0000	

[1] Please refer to the register definition.

The FlexCAN module stores CAN messages for transmission and reception using a message buffer structure. Each MB is formed by 16 bytes mapped in memory as described in Table 29-2. The FlexCAN module can manage as many as 64 message buffers. Table 29-2 shows a standard/extended message buffer (MB0) memory map, using 16 bytes (0x80–0x8F) total space.

Table 29-2. Message Buffer MB0 Memory Mapping

Address Offset	MB Field
0x80	Control and status (C/S)
0x84	Identifier field
0x88–0x8F	Data fields 0–7 (1 byte each)

NOTE

Reading the C/S word of a message buffer (the first word of each MB) locks it, preventing it from receiving further messages until it is unlocked either by reading another MB or by reading the timer.

NOTE

During CAN messages reception by FlexCAN, the RXGMASK (Rx Global Mask) is used as acceptance mask for most of the Rx Message Buffers (MB). When the FIFO Enable bit in the FlexCAN Module Configuration Register (CANx_MCR[FEN], bit 2) is set, the RXGMASK also applies to most of the elements of the ID filter table. However there is a misalignment between the position of the ID field in the Rx MB and in RXIDA, RXIDB, and RXIDC fields of the ID Tables. In fact RXIDA filter in the ID Tables is shifted one bit to the left from Rx MBs ID position as shown below:

- Rx MB ID = bits 3-31 of ID word corresponding to message ID bits 0-28
- RXIDA = bits 2-30 of ID Table corresponding to message ID bits 0-28

Note that the mask bits one-to-one correspondence occurs with the filters bits, not with the incoming message ID bits. This leads the RXGMASK to affect Rx MB and Rx FIFO filtering in different ways. For example, if the user intends to mask out the bit 24 of the ID filter of Message Buffers then the RXGMASK will be configured as 0xffff_ffef. As result, bit 24 of the ID field of the incoming message will be ignored during filtering process for Message Buffers. This very same configuration of RXGMASK would lead bit 24 of RXIDA to be "don't care" and thus bit 25 of the ID field of the incoming message would be ignored during filtering process for Rx FIFO. Similarly, both RXIDB and RXIDC filters have multiple misalignments with regards to position of ID field in Rx MBs, which can lead to erroneous masking during filtering process for either Rx FIFO or MBs. RX14MASK (Rx 14 Mask) and RX15MASK (Rx 15 Mask) have the same structure as the RXGMASK. This includes the misalignment problem between the position of the ID field in the Rx MBs and in RXIDA, RXIDB, and RXIDC fields of the ID Tables.

One of the following actions be taken to avoid the above problems:

- Do not enable the RxFIFO. If CANx_MCR[FEN]=0 then the Rx FIFO is disabled and thus the masks RXGMASK, RX14MASK and RX15MASK do not affect it.
- Enable Rx Individual Mask Registers. If the Backwards Compatibility Configuration bit in the FlexCAN Module Configuration Register (CANx_MCR[BCC], bit 15) is set then the Rx Individual Mask Registers (RXIMR0-63) are enabled and thus the masks RXGMASK, RX14MASK and RX15MASK are not used.
- Do not use masks RXGMASK, RX14MASK and RX15MASK (rather set them to reset value, which is 0xffff_ffff) when CANx_MCR[FEN]=1 and CANx_MCR[BCC]=0. In this case, filtering processes for both Rx MBs and Rx FIFO are not affected by those masks.
- Do not configure any MB as Rx (i.e. let all MBs as either Tx or inactive) when CANx_MCR[FEN]=1 and CANx_MCR[BCC]=0. In this case, the masks RXGMASK, RX14MASK and RX15MASK can be used to affect ID Tables without affecting filtering process for Rx MBs.

29.3.2 Message Buffer Structure

The message buffer structure used by the FlexCAN module is represented in Figure 29-2. Both extended and standard frames (29-bit identifier and 11-bit identifier, respectively) used in the CAN specification (version 2.0 Part B) are represented.

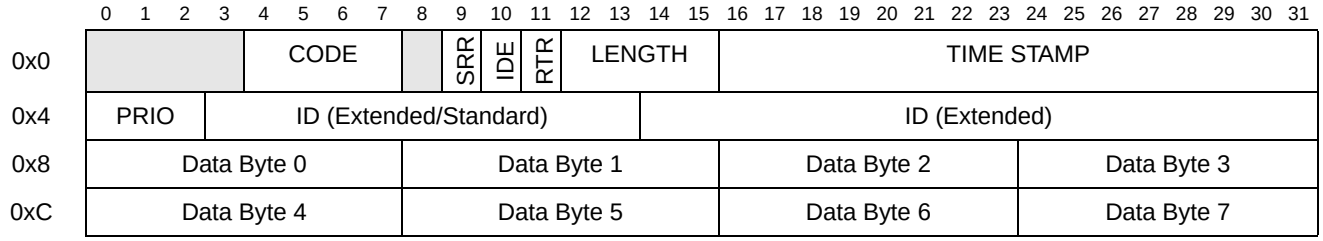


Figure 29-2. Message Buffer Structure

Table 29-3. Message Buffer Field Descriptions

Name	Description
CODE	Message Buffer Code. This 4-bit field can be accessed (read or write) by the CPU and by the FlexCAN module itself, as part of the message buffer matching and arbitration process. The encoding is shown in Table 29-4 and Table 29-5. See Section 29.4, Functional Description, for additional information.
SRR	Substitute Remote Request. Fixed recessive bit, used only in extended format. It must be set to 1 by the user for transmission (Tx Buffers) and is stored with the value received on the CAN bus for Rx receiving buffers. It can be received as either recessive or dominant. If FlexCAN receives this bit as dominant, then it is interpreted as arbitration loss. 0 Dominant is not a valid value for transmission in extended format frames. 1 Recessive value is compulsory for transmission in extended format frames.

Table 29-3. Message Buffer Field Descriptions (continued)

Name	Description
IDE	ID Extended Bit. This bit identifies whether the frame format is standard or extended. 0 Frame format is standard. 1 Frame format is extended.
RTR	Remote Transmission Request. This bit is used for requesting transmissions of a data frame. If FlexCAN transmits this bit as 1 (recessive) and receives it as 0 (dominant), it is interpreted as arbitration loss. If this bit is transmitted as 0 (dominant), then if it is received as 1 (recessive), the FlexCAN module treats it as bit error. If the value received matches the value transmitted, it is considered as a successful bit transmission. 0 Indicates the current MB has a data frame to be transmitted. 1 Indicates the current MB has a remote frame to be transmitted.
LENGTH	Length of Data in Bytes. This 4-bit field is the length (in bytes) of the Rx or Tx data, which is located in offset 0x8 through 0xF of the MB space (see Figure 29-2). In reception, this field is written by the FlexCAN module, copied from the DLC (data length code) field of the received frame. In transmission, this field is written by the CPU and corresponds to the DLC field value of the frame to be transmitted. When RTR = 1, the Frame to be transmitted is a remote frame and does not include the data field, regardless of the LENGTH field.
TIME STAMP	Free-Running Counter Time Stamp. This 16-bit field is a copy of the free-running timer, captured for Tx and Rx frames at the time when the beginning of the Identifier field appears on the CAN bus.
PRIO	Local Priority. This 3-bit field is only used when LPRIO_EN bit is set in CAN*x*_MCR and it only makes sense for Tx buffers. These bits are not transmitted. They are appended to the regular ID to define the transmission priority. See Section 29.4.2, Arbitration Process.
ID	Frame Identifier. In standard frame format, only the 11 most significant bits (3 to 13) are used for frame identification in both receive and transmit cases. The 18 least significant bits are ignored. In extended frame format, all bits are used for frame identification in both receive and transmit cases.
DATA	Data Field. A many as 8 bytes can be used for a data frame. For Rx frames, the data is stored as it is received from the CAN bus. For Tx frames, the CPU prepares the data field to be transmitted within the frame.

Table 29-4. Message Buffer Code for Rx Buffers

Rx Code before Rx New Frame	Description	Rx Code after Rx New Frame	Comment
0000	NOT ACTIVE: MB is not active.	—	MB does not participate in the matching process.
0100	EMPTY: MB is active and empty.	0010	MB participates in the matching process. When a frame is received successfully, the code is automatically updated to FULL.
0010	FULL: MB is full.	0010	The act of reading the C/S word followed by unlocking the MB does not make the code return to EMPTY. It remains FULL. If a new frame is written to the MB after the C/S word was read and the MB was unlocked, the code still remains FULL.
		0110	If the MB is FULL and a new frame is overwritten to this MB before the CPU had time to read it, the code is automatically updated to OVERRUN. Refer to Section 29.4.4, Matching Process, for details about overrun behavior.

Table 29-4. Message Buffer Code for Rx Buffers (continued)

Rx Code before Rx New Frame	Description	Rx Code after Rx New Frame	Comment
0110	OVERRUN: A frame was overwritten into a full buffer.	0010	If the code indicates OVERRUN but the CPU reads the C/S word and then unlocks the MB, when a new frame is written to the MB the code returns to FULL.
		0110	If the code already indicates OVERRUN, and yet another new frame must be written, the MB is overwritten again, and the code remains OVERRUN. Refer to Section 29.4.4, Matching Process, for details about overrun behavior.
0XY1[1]	BUSY: FlexCAN is updating the contents of the MB. The CPU must not access the MB.	0010	An EMPTY buffer was written with a new frame (XY was 01).
		0110	A FULL/OVERRUN buffer was overwritten (XY was 11).

1 Note that for Tx MBs (see Table 29-5), the BUSY bit should be ignored on read, except when AEN bit is set in the CAN*x*_MCR.

Table 29-5. Message Buffer Code for Tx Buffers

RTR	Initial Tx Code	Code after Successful Transmission	Description
X	1000	—	INACTIVE: MB does not participate in the arbitration process.
X	1001	—	ABORT: MB was configured as Tx and CPU aborted the transmission. This code is only valid when AEN bit in CAN*x*_MCR is asserted. MB does not participate in the arbitration process.
0	1100	1000	Transmit data frame unconditionally once. After transmission, the MB automatically returns to the INACTIVE state.
1	1100	0100	Transmit remote frame unconditionally once. After transmission, the MB automatically becomes and Rx MB with the same ID.
0	1010	1010	Transmit a data frame whenever a remote request frame with the same ID is received. This MB participates simultaneously in both the matching and arbitration processes. The matching process compares the ID of the incoming remote request frame with the ID of the MB. If a match occurs this MB is allowed to participate in the current arbitration process and the CODE field is automatically updated to 1110 to allow the MB to participate in future arbitration runs. When the frame is eventually transmitted successfully, the Code automatically returns to 1010 to restart the process again.
0	1110	1010	The MBM generates this code as a result of match to a remote request frame. The data frame is transmitted unconditionally once and then the code automatically returns to ‘1010’. The CPU can also write this code with the same effect.

29.3.3 Rx FIFO Structure

When the FEN bit is set in the CAN*x*_MCR, the memory area from 0x80 to 0xFC(which is normally occupied by MBs 0 to 7) is used by the reception FIFO engine. Figure 29-3 shows the Rx FIFO data structure. The region 0x80–0x8C contains an MB structure which is the port through which the CPU reads

data from the FIFO (the oldest frame received and not read yet). The region 0x90–0xDC is reserved for internal use of the FIFO engine. The region 0xE0–0xFC contains an eight-entry ID table that specifies filtering criteria for accepting frames into the FIFO. Figure 29-4 shows the three different formats that the elements of the ID table can assume, depending on the IDAM field of the CAN*x*_MCR. Note that all elements of the table must have the same format. See Section 29.4.6, Rx FIFO, for more information.

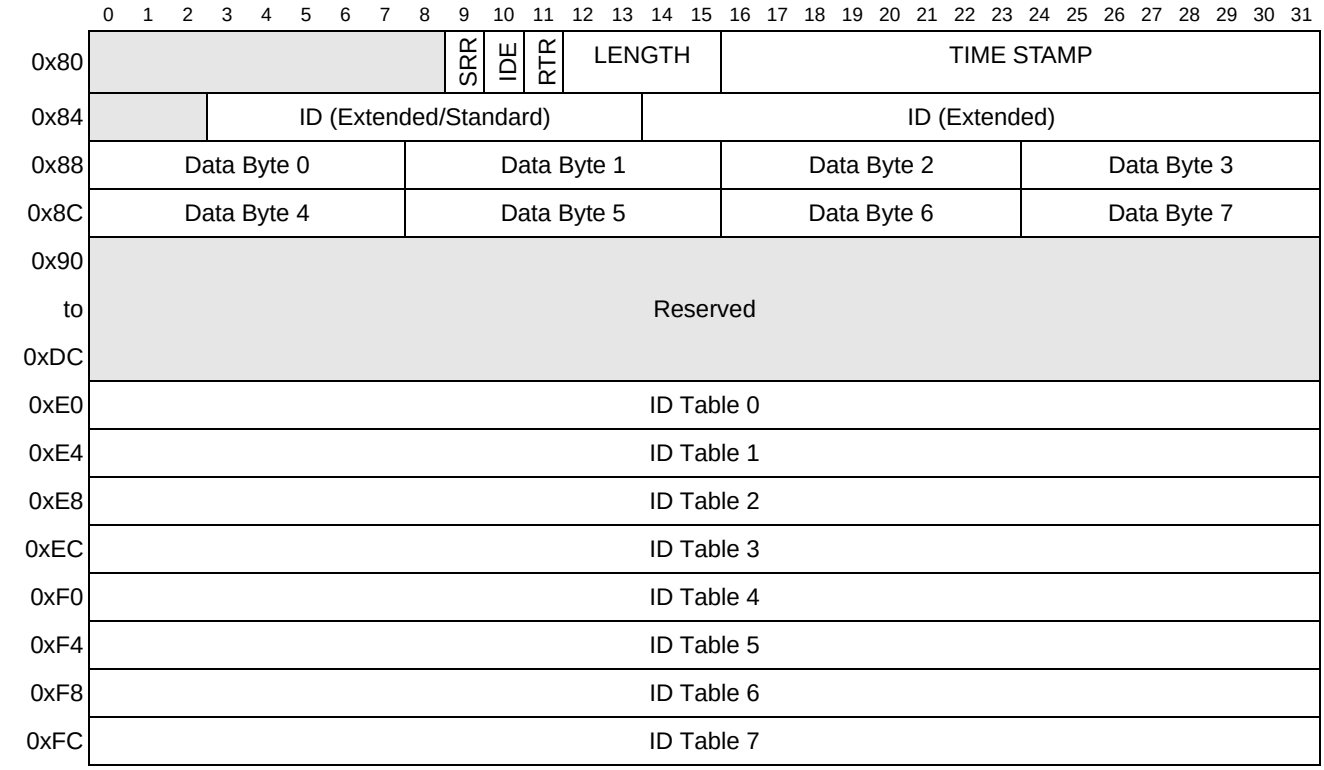


Figure 29-3. Rx FIFO Structure

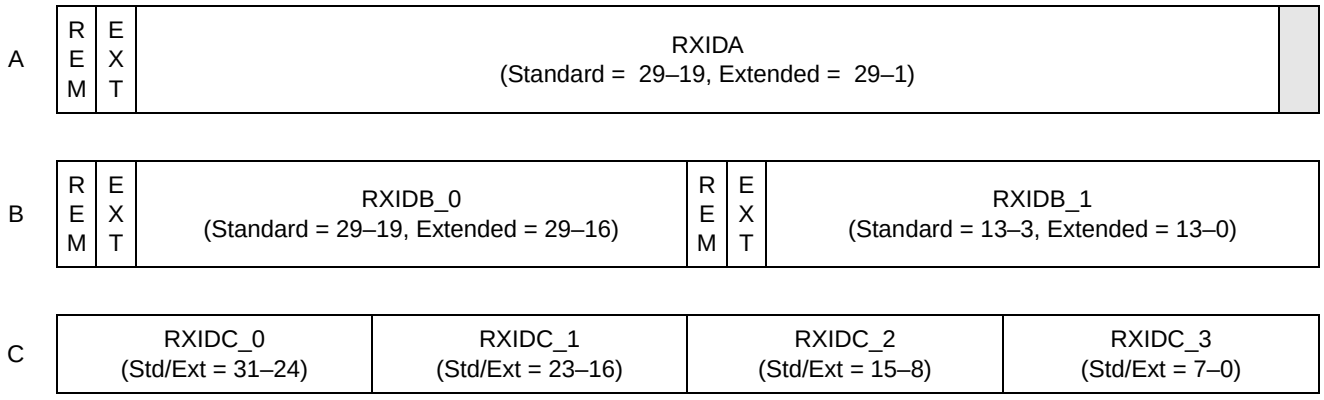


Figure 29-4. ID Table 0–7

Table 29-6. ID Table 0–7 Field Descriptions

Name	Description
REM	Remote Frame. This bit specifies if Remote Frames are accepted into the FIFO if they match the target ID. 0 Remote frames are rejected and data frames can be accepted. 1 Remote frames can be accepted and data frames are rejected.
EXT	Extended Frame. Specifies whether extended or standard frames are accepted into the FIFO if they match the target ID. 0 Extended frames are rejected and standard frames can be accepted. 1 Extended frames can be accepted and standard frames are rejected.
RXIDA	Rx Frame Identifier (Format A). Specifies an ID to be used as acceptance criteria for the FIFO. In the standard frame format, only the 11 most significant bits (3 to 13) are used for frame identification. In the extended frame format, all bits are used.
RXIDB_0 RXIDB_1	Rx Frame Identifier (Format B). Specifies an ID to be used as acceptance criteria for the FIFO. In the standard frame format, the 11 most significant bits (a full standard ID) (3 to 13) are used for frame identification. In the extended frame format, all 14 bits of the field are compared to the 14 most significant bits of the received ID.
RXIDC_0 RXIDC_1 RXIDC_2 RXIDC_3	Rx Frame Identifier (Format C). Specifies an ID to be used as acceptance criteria for the FIFO. In both standard and extended frame formats, all 8 bits of the field are compared to the 8 most significant bits of the received ID.

29.3.4 Register Descriptions

This section lists the FlexCAN registers in address order and describes the registers and their bit fields.

29.3.4.1 Module Configuration Register (CAN*x*_MCR)

This register defines global system configurations, such as the module operation mode (e.g., low power) and maximum message buffer configuration. Most of the fields in this register can be accessed at any time, except the MAXMB field, which should only be changed while the module is in freeze mode.

Offset: Base + 0x0000 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	MDIS	FRZ	FEN	HALT	NOT_RDY	0	SOFT_RST	FRZ_ACK	1	0	WRN_EN	LPM_ACK	0	0	SRX_DIS	BCC
W																
Reset	1	1	0	1	1	0	0	0	1[1]	0	0	1	0[1]	0[1]	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	LPRIO_EN	AEN	0	0	IDAM		0	0	MAXMB					
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1

1 Writes to this bit have no effect, but reads return the written value.

Figure 29-5. Module Configuration Register (CAN*x*_MCR)

Table 29-7. CAN*x*_MCR Field Descriptions

Field	Description
MDIS	Module Disable. Controls whether FlexCAN is enabled or not. When disabled, FlexCAN shuts down the clock to the CAN protocol interface and message buffer management submodules. This is the only bit in CAN*x*_MCR not affected by soft reset. See Section 29.4.8.2, Module Disabled Mode, for more information. 0 Enable the FlexCAN module. 1 Disable the FlexCAN module.
FRZ	Freeze Enable. Specifies the FlexCAN behavior when the HALT bit in the CAN*x*_MCR is set or when debug mode is requested at MCU level. When FRZ is set, FlexCAN is enabled to enter freeze mode. Clearing this bit causes FlexCAN to exit from freeze mode. 0 Not enabled to enter freeze mode. 1 Enabled to enter freeze mode.
FEN	FIFO Enable. controls whether the FIFO feature is enabled or not. When FEN is set, MBs 0 to 7 cannot be used for normal reception and transmission because the corresponding memory region (0x80–0xFF) is used by the FIFO engine. See Section 29.3.3, Rx FIFO Structure, and Section 29.4.6, Rx FIFO, for more information. 0 FIFO not enabled. 1 FIFO enabled.
HALT	Halt FlexCAN. Assertion of this bit puts the FlexCAN module into freeze mode if FRZ is asserted. The CPU clears it after initializing the message buffers and CAN*x*_CTRL. If FRZ is set, no reception or transmission is performed by FlexCAN before this bit is cleared. While in freeze mode, the CPU has write access to the CAN*x*_ECR, that is otherwise read-only. Freeze mode cannot be entered while FlexCAN is disabled. See Section 29.4.8.1, Freeze Mode, for more information. 0 No freeze mode request. 1 Enters freeze mode if the FRZ bit is asserted.
NOT_RDY	FlexCAN Not Ready. Indicates that FlexCAN is either disabled or in freeze mode. This flag is cleared once FlexCAN has exited these modes. 0 FlexCAN module is in normal mode, listen-only mode, or loop-back mode. 1 FlexCAN module is either disabled or in freeze mode.

Table 29-7. CAN*x*_MCR Field Descriptions (continued)

Field	Description
SOFT_RST	Soft Reset. When asserted, FlexCAN resets its internal state machines and some of the memory-mapped registers. The following registers are affected by soft reset: • CAN*x*_MCR (except the MDIS bit) • CAN*x*_TIMER • CAN*x*_ECR • CAN*x*_ESR • CAN*x*_IMASK1 • CAN*x*_IMASK2 • CAN*x*_IFLAG1 • CAN*x*_IFLAG2 Configuration registers that control the interface to the CAN bus are not affected by soft reset. The following registers are unaffected: • CAN*x*_CTRL • CAN*x*_RXGMASK • CAN*x*_RX14MASK • CAN*x*_RX15MASK • All message buffers The SOFT_RST bit can be asserted directly by the CPU when it writes to the CAN*x*_MCR, but it is also asserted when global soft reset is requested at MCU level. Because soft reset is synchronous and has to follow a request/acknowledge procedure across clock domains, it may take some time to fully propagate its effect. The SOFT_RST bit remains asserted while reset is pending, and is automatically negated when reset completes. Therefore, software can poll this bit to know when the soft reset has completed. 0 No reset request. 1 Resets values in registers indicated above.
FRZ_ACK	Freeze Mode Acknowledge. Indicates that FlexCAN is in freeze mode and its prescaler is stopped. The freeze mode request cannot be granted until current transmission and reception processes have finished. Therefore the software can poll the FRZ_ACK bit to know when FlexCAN has actually entered freeze mode. If freeze mode request is negated, then this bit is negated once the FlexCAN prescaler is running again. If freeze mode is requested while FlexCAN is disabled, then the FRZ_ACK bit is only set when the low-power mode is exited. See Section 29.4.8.1, Freeze Mode, for more information. 0 FlexCAN not in freeze mode, prescaler running. 1 FlexCAN in freeze mode, prescaler stopped.
WRN_EN	Warning Interrupt Enable. When set, this bit enables the generation of the TWRN_INT and RWRN_INT flags in the error and status register. If WRN_EN is negated, the TWRN_INT and RWRN_INT flags is always 0, independent of the values of the error counters, and no warning interrupt is ever generated. 1 TWRN_INT and RWRN_INT bits are set when the respective error counter transition from < 96 to ≥ 96. 0 TWRN_INT and RWRN_INT bits are 0, independent of the values in the error counters.
LPM_ACK	Low-Power Mode Acknowledge. Indicates whether FlexCAN is disabled. This cannot be performed until all current transmission and reception processes have finished, so the CPU can poll the LPM_ACK bit to know when FlexCAN has actually been disabled. See Section 29.4.8.2, Module Disabled Mode, for more information. 0 FlexCAN not disabled. 1 FlexCAN is disabled.
SRX_DIS	Self Reception Disable. This bit defines whether FlexCAN is allowed to receive frames transmitted by itself. If this bit is set, frames transmitted by the module are not stored in any MB, regardless if the MB is programmed with an ID that matches the transmitted frame, and no interrupt flag or interrupt signal is generated due to the frame reception. 0 Self reception enabled. 1 Self reception disabled.

Table 29-7. CAN*x*_MCR Field Descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>BCC</td><td>Backwards Compatibility Configuration. Provided to support backwards compatibility with previous FlexCAN versions. When this bit is negated, the following configuration is applied:
• For MCUs supporting individual Rx ID masking, this feature is disabled. Instead of individual ID masking per MB, FlexCAN uses its previous masking scheme with CANx_RXGMASK, CANx_RX14MASK, and CANx_RX15MASK.
• The reception queue feature is disabled. On receiving a message, if the first MB with a matching ID that is found is still occupied by a previous unread message, FlexCAN does not look for another matching MB. It overrides this MB with the new message and sets the CODE field to 0b0110 (overrun).
This bit is cleared on reset, allowing legacy software to work without modification.
0 Individual Rx masking and queue feature are disabled.
1 Individual Rx masking and queue feature are enabled.</td></tr>
<tr><td>LPRIO_EN</td><td>Local Priority Enable. Provided for backwards compatibility reasons. It controls whether the local priority feature is enabled or not. It is used to extend the ID used during the arbitration process. With this extended ID concept, the arbitration process is done based on the full 32-bit word, but the actual transmitted ID still has 11 bits for standard frames and 29 bits for extended frames.
0 Local priority disabled.
1 Local priority enabled.</td></tr>
<tr><td>AEN</td><td>Abort Enable. Provided for backwards compatibility reasons. When set, it enables the Tx abort feature. This feature guarantees a safe procedure for aborting a pending transmission, so that no frame is sent in the CAN bus without notification.
0 Abort disabled.
1 Abort enabled.</td></tr>
<tr><td>IDAM</td><td>ID Acceptance Mode. Identifies the format of the elements of the Rx FIFO filter table. All elements of the table are configured at the same time by this field (they are all the same format).
<table>
<tr><th>IDAM</th><th>Format</th><th>Explanation</th></tr>
<tr><td>00</td><td>A</td><td>One full ID (standard or extended) per filter element.</td></tr>
<tr><td>01</td><td>B</td><td>Two full standard IDs or two partial 14-bit extended IDs per filter element.</td></tr>
<tr><td>10</td><td>C</td><td>Four partial 8-bit IDs (standard or extended) per filter element.</td></tr>
<tr><td>11</td><td>D</td><td>All frames rejected.</td></tr>
</table></td></tr>
<tr><td>MAXMB</td><td>Maximum Number Of Message Buffers. This 6-bit field defines the maximum number of message buffers of the FlexCAN module. The reset value (0x0F) is equivalent to a 16 MB configuration. This field should be changed only while the module is in freeze mode.
Maximum MBs in use = MAXMB + 1
Note: MAXMB must be programmed with a value smaller or equal to the number of available message buffers, otherwise FlexCAN can transmit and receive wrong messages.</td></tr>
</table>

29.3.4.2 Control Register (CAN*x*_CTRL)

This register is defined for specific FlexCAN control features related to the CAN bus, such as bit-rate, programmable sampling point within an Rx bit, loop back mode, listen only mode, bus off recovery behavior, and interrupt enabling (bus-off, error, warning). It also determines the division factor for the clock prescaler. Most of the fields in this register should only be changed while the module is disabled or

in freeze mode. Exceptions are the BOFF_MSK, ERR_MSK, TWRN_MSK, RWRN_MSK, and BOFF_REC bits, which can be accessed at any time.

Offset: Base + 0x0004 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	PRESDIV								RJW		PSEG1			PSEG2		
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17v	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	BOFF_MSK	ERR_MSK	CLK_SRC	LPB	TWRN_MSK	RWRN_MSK	0	0	SMP	BOFF_REC	TSYN	LBUF	LOM	PROPSEG		
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-6. Control Register (CAN*x*_CTRL)

Table 29-8. CAN*x*_CTRL Field Descriptions

Bits	Description
PRESDIV	Prescaler Division Factor. Defines the ratio between the CPI clock frequency and the serial clock (SCK) frequency. The SCK period defines the time quantum of the CAN protocol. For the reset value, the SCK frequency is equal to the CPI clock frequency. The maximum value of this register is 0xFF, that gives a minimum SCK frequency equal to the CPI clock frequency divided by 256. For more information, refer to Section 29.4.7.4, Protocol Timing. $\text{S-clock frequency} = \frac{\text{CPI clock frequency}}{\text{PRESDIV} + 1}$
RJW	Resync Jump Width. Defines the maximum number of time quanta that a bit time can be changed by one re-synchronization. One time quantum is equal to one SCK period. The valid programmable values are 0–3. $\text{Resync Jump Width} = \text{RJW} + 1$
PSEG	Phase Segment 1. Defines the length of phase buffer segment 1 in the bit time. The valid programmable values are 0–7. $\text{Phase Buffer Segment 1} = (\text{PSEG1} + 1) \times \text{Time Quanta}$
PSEG2	Phase Segment 2. Defines the length of phase buffer segment 2 in the bit time. The valid programmable values are 1–7. $\text{Phase Buffer Segment 2} = (\text{PSEG2} + 1) \times \text{Time Quanta}$
BOFF_MSK	Bus Off Mask. Provides a mask for the bus off interrupt. 0 Bus off interrupt disabled. 1 Bus off interrupt enabled.
ERR_MSK	Error Mask. Provides a mask for the error interrupt. 0 Error interrupt disabled. 1 Error interrupt enabled.

Table 29-8. CAN*x*_CTRL Field Descriptions

Bits	Description
CLK_SRC	CAN Engine Clock Source. Selects the clock source to the CAN Protocol Interface (CPI) to be either the system clock (driven by the PLL) or the crystal oscillator clock. The selected clock is the one fed to the prescaler to generate the serial clock (SCK). In order to guarantee reliable operation, this bit should only be changed while the module is disabled. 0 The CAN engine clock source is the oscillator clock. 1 The CAN engine clock source is the system clock.
LPB	Loop Back. Configures FlexCAN to operate in loop-back mode. See Section 29.4.8, Modes of Operation Details, for information about this operating mode. 0 Loop back disabled. 1 Loop back enabled.
TWRN_MSK	This bit provides a mask for the Tx Warning Interrupt associated with the TWRN_INT flag in the ESR register. This bit has no effect if the WRN_EN bit in CAN*x*_MCR is negated and it is read as zero when WRN_EN is negated. 1 Tx Warning Interrupt enabled. 0 Tx Warning Interrupt disabled.
RWRN_MSK	This bit provides a mask for the RX Warning Interrupt associated with the RWRN_INT flag in the Error and Status Register. This bit has no effect if the WRN_EN bit in CAN*x*_MCR is negated and it is read as zero when WRN_EN is negated. 1 Rx Warning Interrupt enabled. 0 Rx Warning Interrupt disabled.
SMP	Sampling Mode. Defines the sampling mode of each bit in the receiving messages (Rx). 0 Just one sample is used to determine the Rx bit value. 1 Three samples are used to determine the value of the received bit: the regular one (sample point) and 2 preceding samples, a majority rule is used.
BOFF_REC	Bus Off Recovery Mode. Defines how FlexCAN recovers from bus off state. If this bit is negated, automatic recovering from bus off state occurs according to the CAN Specification 2.0B. If this bit is set, automatic recovering from bus off is disabled and the module remains in bus off state until the bit is negated by the user. If the negation occurs before 128 sequences of 11 recessive bits are detected on the CAN bus, then bus off recovery happens as if the BOFF_REC bit had never been asserted. If the negation occurs after 128 sequences of 11 recessive bits occurred, then FlexCAN re-synchronizes to the bus by waiting for 11 recessive bits before joining the bus. After negation, the BOFF_REC bit can be re-asserted again during bus off, but it is only effective the next time the module enters bus off. If BOFF_REC was negated when the module entered bus off, asserting it during bus off is not effective for the current bus off recovery. 0 Automatic recovering from bus off state enabled, according to CAN Spec 2.0 part B. 1 Automatic recovering from bus off state disabled.
TSYN	Timer Sync Mode. Enables a mechanism that resets the free-running timer each time a message is received in message buffer 0. This feature provides means to synchronize multiple FlexCAN stations with a special SYNC message (that is, global network time). If the FEN bit in CAN*x*_MCR is set (FIFO enabled), MB8 is used for timer synchronization instead of MB0. 0 Timer sync feature disabled. 1 Timer sync feature enabled. **Note:** There is a possibility of 4–5 ticks count skew between the different FlexCAN stations that would operate in this mode.
LBUF	Lowest Buffer Transmitted First. This bit defines the ordering mechanism for message buffer transmission.When asserted, the LPRIO_EN bit does not affect the priority arbitration. 0 Buffer with highest priority is transmitted first. 1 Lowest number buffer is transmitted first.

Table 29-8. CAN*x*_CTRL Field Descriptions

Bits	Description
LOM	Listen-Only Mode. Configures FlexCAN to operate in listen-only mode. In this mode, the FlexCAN module receives messages without giving any acknowledge. It is not possible to transmit any message in this mode. 0 FlexCAN module is in normal active operation, listen only mode is deactivated. 1 FlexCAN module is in listen only mode operation.
PROPSEG	Propagation Segment. Defines the length of the propagation segment in the bit time. The valid programmable values are 0–7. Propagation Segment Time = (PROPSEG + 1) × Time Quanta Time Quantum = one S clock period

29.3.4.3 Free-Running Timer (CAN*x*_TIMER)

This register represents a 16-bit free-running counter that can be read and written by the CPU. The timer starts from 0x0000 after Reset, counts linearly to 0xFFFF, and wraps around.

The timer is clocked by the FlexCAN bit-clock (which defines the baud rate on the CAN bus). During a message transmission/reception, it increments by one for each bit that is received or transmitted. When there is no message on the bus, it counts using the previously programmed baud rate. During Freeze Mode, the timer is not incremented.

The timer value is captured at the beginning of the identifier field of any frame on the CAN bus. This captured value is written into the Time Stamp entry in a message buffer after a successful reception or transmission of a message.

Writing to the timer is an indirect operation. The data is first written to an auxiliary register and then an internal request/acknowledge procedure across clock domains is executed. All this is transparent to the user, except for the fact that the data takes some time to be actually written to the register. If desired, software can poll the register to discover when the data was actually written.

Offset: Base + 0x0008 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	TIMER															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-7. Free-Running Timer (CAN*x*_TIMER)

29.3.4.4 Rx Mask Registers

By negating the CAN*x*_MCR[BCC] bit, the CAN*x*_RXGMASK, CAN*x*_RX14MASK, and CAN*x*_RX15MASK registers are used as acceptance masks for received frame IDs. Three masks are

defined: a global mask, used for Rx buffers 0–13 and 16–63, and two extra masks dedicated for buffers 14 and 15. The meaning of each mask bit is the following:

- Mask bit = 0: the corresponding incoming ID bit is “don’t care.”
- Mask bit = 1: the corresponding ID bit is checked against the incoming ID bit, to see if a match exists.

Note that these masks are used both for standard and extended ID formats. The value of mask registers should not be changed while in normal operation. Locked frames which had matched a MB through a mask may be transferred into the MB (upon release) but may no longer match. Table 29-9 shows some examples of ID masking for standard and extended message buffers.

Table 29-9. Mask Examples for Standard/Extended Message Buffers

	Base ID ID28.....ID18	IDE	Extended ID ID17.....ID0	Match
MB2 ID	1 1 1 1 1 1 1 1 0 0 0	0		
MB3 ID	1 1 1 1 1 1 1 1 0 0 0	1	0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1	
MB4 ID	0 0 0 0 0 0 0 1 1 1 1	0		
MB5 ID	0 0 0 0 0 0 0 1 1 1 0 1	1	0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1	
MB14 ID	1 1 1 1 1 1 1 1 0 0 0	1	0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1	
Rx Global Mask	1 1 1 1 1 1 1 1 1 1 0		1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 1	
Rx Msg in[1]	1 1 1 1 1 1 1 1 0 0 1	1	0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1	3
Rx Msg in[2]	1 1 1 1 1 1 1 1 0 0 1	0		2
Rx Msg in[3]	1 1 1 1 1 1 1 1 0 0 1	1	0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 0	
Rx Msg in[4]	0 1 1 1 1 1 1 1 0 0 0	0		
Rx Msg in[5]	0 1 1 1 1 1 1 1 0 0 0	1	0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1	14
Rx 14 Mask	0 1 1 1 1 1 1 1 1 1 1		1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0	
Rx Msg in[6]	1 0 1 1 1 1 1 1 0 0 0	1	0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1	
Rx Msg in[7]	0 1 1 1 1 1 1 1 0 0 0	1	0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1	14

1 Match for Extended Format (MB3).
2 Match for Standard Format. (MB2).
3 Mismatch for MB3 because of ID0.
4 Mismatch for MB2 because of ID28.
5 Mismatch for MB3 because of ID28, Match for MB14 (uses CAN*x*_RX14MASK).
6 Mismatch for MB14 because of ID27 (uses CAN*x*_RX14MASK).
7 Match for MB14 (uses CAN*x*_RX14MASK).

29.3.4.4.1 Rx Global Mask (CAN*x*_RXGMASK)

This register is provided for legacy support. On the PXN20, setting the BCC bit in CAN*x*_MCR causes the CAN*x*_RXGMASK Register to have no effect on the module operation.

CAN*x*_RXGMASK is used as acceptance mask for all Rx MBs, excluding MBs 14–15, which have individual mask registers. When the FEN bit in CAN*x*_MCR is set (FIFO enabled), the

CAN*x*_RXGMASK also applies to all elements of the ID filter table, except elements 6-7, which have individual masks.

The contents of this register must be programmed while the module is in Freeze Mode, and must not be modified when the module is transmitting or receiving frames.

Offset: Base + 0x0010 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	MI31	MI30	MI29	MI28	MI27	MI26	MI25	MI24	MI23	MI22	MI21	MI20	MI19	MI18	MI17	MI16
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	MI15	MI14	MI13	MI12	MI11	MI10	MI9	MI8	MI7	MI6	MI5	MI4	MI3	MI2	MI1	MI0
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Figure 29-8. Rx Mask Register (CAN*x*_RXGMASK)

Table 29-10. CAN*x*_RXGMASK Field Descriptions

Field	Description
MI*n*	Mask Bits. For normal Rx MBs, the mask bits affect the ID filter programmed on the MB. For the Rx FIFO, the mask bits affect all bits programmed in the filter table (ID, IDE, RTR). 0 the corresponding bit in the filter is “don’t care.” 1 The corresponding bit in the filter is checked against the one received.

29.3.4.4.2 Rx 14 Mask (CAN*x*_RX14MASK)

This register is provided for legacy support. On the PXN20, setting the BCC bit in CAN*x*_MCR causes the CAN*x*_RX14MASK register to have no effect on the module operation.

CAN*x*_RX14MASK is used as acceptance mask for the Identifier in Message Buffer 14. When the FEN bit in CAN*x*_MCR is set (FIFO enabled), the CAN*x*_RX14MASK also applies to element 6 of the ID filter table. This register has the same structure as the Rx Global Mask Register. It must be programmed while the module is in freeze mode, and must not be modified when the module is transmitting or receiving frames.

- Address Offset: 0x14
- Reset Value: 0xFFFF_FFFF

29.3.4.4.3 Rx 15 Mask (CAN*x*_RX15MASK)

This register is provided for legacy support. On the PXN20, setting the BCC bit in CAN*x*_MCR causes the CAN*x*_RX15MASK register to have no effect on the module operation.

When the BCC bit is negated, CAN*x*_RX15MASK is used as acceptance mask for the Identifier in Message Buffer 15. When the FEN bit in CAN*x*_MCR is set (FIFO enabled), the CAN*x*_RX14MASK also applies to element 7 of the ID filter table. This register has the same structure as the Rx Global Mask Register. It must be programmed while the module is in freeze mode, and must not be modified when the module is transmitting or receiving frames.

- Address Offset: 0x18
- Reset Value: 0xFFFF_FFFF

29.3.4.5 Error Counter Register (CAN*x*_ECR)

CAN*x*_ECR has two 8-bit fields reflecting the value of two FlexCAN error counters: the transmit error counter (TXECTR field) and receive error counter (RXECTR field). The rules for increasing and decreasing these counters are described in the CAN protocol and are completely implemented in the FlexCAN module. Both counters are read-only except in freeze mode, where they can be written by the CPU.

Writing to the CAN*x*_ECR while in freeze mode is an indirect operation. The data is first written to an auxiliary register and then an internal request/acknowledge procedure across clock domains is executed. All this is transparent to the user, except for the fact that the data takes some time to be actually written to the register. If desired, software can poll the register to discover when the data was actually written.

FlexCAN responds to any bus state as described in the protocol: transmitting, for example, an error active or error passive flag, delaying its transmission start time (error passive), and avoiding any influence on the bus when in the bus off state. The following are the basic rules for FlexCAN bus state transitions:

- If the value of TXECTR or RXECTR increases to be greater than or equal to 128, the FLT_CONF field in the CAN*x*_ESR is updated to reflect the error passive state.
- If the FlexCAN state is error passive, and either TXECTR or RXECTR decrements to a value less than or equal to 127 while the other already satisfies this condition, the FLT_CONF field in the CAN*x*_ESR is updated to reflect the ‘error active’ state.
- If the value of TXECTR increases to be greater than 255, the FLT_CONF field in the CAN*x*_ESR is updated to reflect the bus off state, and an interrupt may be issued. The value of TXECTR is then reset to 0.
- If FlexCAN is in the bus off state, then TXECTR is cascaded together with another internal counter to count the 128th occurrences of 11 consecutive recessive bits on the bus. Hence, TXECTR is reset to 0 and counts in a manner where the internal counter counts 11 such bits and then wraps around while incrementing the TXECTR. When TXECTR reaches the value of 128, the FLT_CONF field in CAN*x*_ESR is updated to be error active, and both error counters are reset to 0. At any instance of dominant bit following a stream of less than 11 consecutive recessive bits, the internal counter resets itself to 0 without affecting the TXECTR value.
- If during system start-up, only one node is operating, then its TXECTR increases in each message it is trying to transmit, as a result of acknowledge errors (indicated by the ACK_ERR bit in CAN*x*_ESR). After the transition to the error passive state, the TXECTR is not incremented by acknowledge errors. Therefore, the device never goes to the bus off state.

If the RXECTR increases to a value greater than 127, it is not incremented further, even if more errors are detected while being a receiver. At the next successful message reception, the counter is set to a value between 119 and 127 to resume the error active state.

Offset: Base + 0x001C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	RXECTR								TXECTR							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-9. Error Counter Register (CAN*x*_ECR)

29.3.4.6 Error and Status Register (CAN*x*_ESR)

This register reflects various error conditions, some general status of the device, and it is the source of four interrupts to the CPU. The reported error conditions (bits 16–21) are those that occurred since the last time the CPU read this register. The CPU read action clears bits 16–23. Bits 22–28 are status bits.

Most bits in this register are read-only, except TWRN_INT, RWRN_INT, BOFF_INT, and ERR_INT, which are interrupt flags that can be cleared by writing 1 to them (writing 0 has no effect).

NOTE

A read clears BIT1_ERR, BIT0_ERR, ACK_ERR, CRC_ERR, FRM_ERR, and STF_ERR, therefore these bits must not be read speculatively.

Offset: Base + 0x0020 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	TWRN_INT	RWRN_INT
W															w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	BIT1_ERR	BIT0_ERR	ACK_ERR	CRC_ERR	FRM_ERR	STF_ERR	TX_WRN	RX_WRN	IDLE	TXRX	FLT_CONF		0	BOFF_INT	ERR_INT	0
W	r1c	r1c	r1c	r1c	r1c	r1c	r1c	r1c						w1c	w1c	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-10. Error and Status Register (CAN*x*_ESR)

Table 29-11. CAN*x*_ESR Field Descriptions

Field	Description
TWRN_INT	If the WRN_EN bit in CAN*x*_MCR is set, the TWRN_INT bit is set when the TXWRN flag transitions from 0 to 1, meaning that the Tx error counter reached 96. If the corresponding mask bit in the CAN*x*_CTRL register (TWRN_MSK) is set, an interrupt is generated to the CPU. This bit is cleared by writing to 1. Writing 0 has no effect. 0 No such occurrence. 1 TXECTR ≥ 96.
RWRN_INT	If the WRN_EN bit in CAN*x*_MCR is set, the RWRN_INT bit is set when the RXWRN flag transitions from 0 to 1, meaning that the Rx error counter reached 96. If the corresponding mask bit in the CAN*x*_CTRL register (RWRNMSK) is set, an interrupt is generated to the CPU. This bit is cleared by writing to 1. Writing 0 has no effect. 0 No such occurrence. 1 RXECTR ≥ 96.
BIT1_ERR	Bit 1 Error. Indicates when an inconsistency occurs between the transmitted and the received message in a bit. A read clears BIT1_ERR. 0 No such occurrence. 1 At least one bit sent as recessive is received as dominant.This bit is not set by a transmitter in case of arbitration field or ACK slot, or in case of a node sending a passive error flag that detects dominant bits. **Note:** This bit is not set by a transmitter in case of arbitration field or ACK slot, or in case of a node sending a passive error flag that detects dominant bits.
BIT0_ERR	Bit 0 Error. Indicates when an inconsistency occurs between the transmitted and the received message in a bit. A read clears BIT0_ERR. 0 No such occurrence. 1 At least one bit sent as dominant is received as recessive.
ACK_ERR	Acknowledge Error. Indicates that an acknowledge error has been detected by the transmitter node; that is, a dominant bit has not been detected during the ACK SLOT. A read clears ACK_ERR. 0 No such occurrence. 1 An ACK error occurred since last read of this register.
CRC_ERR	Cyclic Redundancy Code Error. Indicates that a CRC error has been detected by the receiver node; that is, the calculated CRC is different from the received. A read clears CRC_ERR. 0 No such occurrence. 1 A CRC error occurred since last read of this register.
FRM_ERR	Form Error. Indicates that a form error has been detected by the receiver node; that is, a fixed-form bit field contains at least one illegal bit. A read clears FRM_ERR. 0 No such occurrence. 1 A form error occurred since last read of this register.
STF_ERR	Stuffing Error. Indicates that a stuffing error has been detected. A read clears STF_ERR. 0 No such occurrence. 1 A stuffing error occurred since last read of this register.
TX_WRN	Tx Error Counter. This status bit indicates that repetitive errors are occurring during message transmission. A read clears TX_WRN. 0 No such occurrence. 1 TXECTR ≥ 96.
RX_WRN	Rx Error Counter. This status bit indicates when repetitive errors are occurring during messages reception. A read clears RX_WRN. 0 No such occurrence. 1 RXECTR ≥ 96.

Table 29-11. CAN*x*_ESR Field Descriptions (continued)

Field	Description
IDLE	CAN Bus IDLE State. This status bit indicates when CAN bus is in IDLE state. 0 No such occurrence. 1 CAN bus is now IDLE.
TXRX	Current FlexCAN Status (Transmitting/Receiving). This status bit indicates if FlexCAN is transmitting or receiving a message when the CAN bus is not in IDLE state. This bit has no meaning when IDLE is asserted. 0 FlexCAN is receiving a message (IDLE = 0). 1 FlexCAN is transmitting a message (IDLE = 0).
FLT_CONF	Fault Confinement State. This status bit indicates the confinement state of the FlexCAN module. If the LOM bit in the CAN*x*_CTRL is asserted, the FLT_CONF field indicates “Error Passive”. Since the CAN*x*_CTRL is not affected by soft reset, the FLT_ CONF field is not affected by soft reset if the LOM bit is asserted. 00 Error active. 01 Error passive. 1*n* Bus off.
BOFF_INT	Bus Off Interrupt. This status bit is set when FlexCAN is in the bus off state. If CAN*x*_CTRL[BOFF_MSK] is set, an interrupt is generated to the CPU. This bit is cleared by writing it to 1. Writing 0 has no effect. 0 No such occurrence. 1 FlexCAN module is in “Bus Off” state.
ERR_INT	Error Interrupt. This status bit indicates that at least one of the error bits (bits 16–21) is set. If CAN*x*_CTRL[ERR_MSK] is set, an interrupt is generated to the CPU. This bit is cleared by writing it to 1. Writing 0 has no effect. 0 No such occurrence. 1 Indicates setting of any error bit in the CANx_ESR.

29.3.4.7 Interrupt Masks 2 Register (CAN*x*_IMASK2)

This register allows any number of a range of 32 message buffer interrupts to be enabled or disabled. It contains one interrupt mask bit per buffer, enabling the CPU to determine which buffer generates an interrupt after a successful transmission or reception (that is, when the corresponding CAN*x*_IFLAG2 bit is set).

Offset: Base + 0x0024 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	BUF 63M	BUF 62M	BUF 61M	BUF 60M	BUF 59M	BUF 58M	BUF 57M	BUF 56M	BUF 55M	BUF 54M	BUF 53M	BUF 52M	BUF 51M	BUF 50M	BUF 49M	BUF 48M
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	BUF 47M	BUF 46M	BUF 45M	BUF 44M	BUF 43M	BUF 42M	BUF 41M	BUF 40M	BUF 39M	BUF 38M	BUF 37M	BUF 36M	BUF 35M	BUF 34M	BUF 33M	BUF 32M
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-11. Interrupt Masks 2 Register (CAN*x*_IMASK2)

Table 29-12. CAN*x*_IMASK2 Field Descriptions

Field	Description
BUF*n*M	Message Buffer *n* Mask. Enables or disables the respective FlexCAN message buffer (MB63 to MB32) Interrupt. 0 The corresponding buffer Interrupt is disabled. 1 The corresponding buffer Interrupt is enabled. **Note:** Setting or clearing a bit in the CAN*x*_IMASK2 register can assert or negate an interrupt request, respectively.

29.3.4.8 Interrupt Masks 1 Register (CAN*x*_IMASK1)

This register allows to enable or disable any number of a range of 32 message buffer interrupts. It contains one interrupt mask bit per buffer, enabling the CPU to determine which buffer generates an interrupt after a successful transmission or reception (i.e., when the corresponding CAN*x*_IFLAG1 bit is set).

Offset: Base + 0x0028 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	BUF 31M	BUF 30M	BUF 29M	BUF 28M	BUF 27M	BUF 26M	BUF 25M	BUF 24M	BUF 23M	BUF 22M	BUF 21M	BUF 20M	BUF 19M	BUF 18M	BUF 17M	BUF 16M
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	BUF 15M	BUF 14M	BUF 13M	BUF 12M	BUF 11M	BUF 10M	BUF 9M	BUF 8M	BUF 7M	BUF 6M	BUF 5M	BUF 4M	BUF 3M	BUF 2M	BUF 1M	BUF 0M
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-12. Interrupt Masks 1 Register (CAN*x*_IMASK1)

Table 29-13. CANx_IMASK1 Field Descriptions

Field	Description
BUF*n*M	Message Buffer *n* Mask. Enables or disables the respective FlexCAN message buffer (MB31 to MB0) Interrupt. 0 The corresponding buffer Interrupt is disabled. 1 The corresponding buffer Interrupt is enabled. **Note:** Setting or clearing a bit in the CAN*x*_IMASK1 register can assert or negate an interrupt request, respectively.

29.3.4.9 Interrupt Flags 2 Register (CAN*x*_IFLAG2)

This register defines the flags for 32 message buffer interrupts. It contains one interrupt flag bit per buffer. Each successful transmission or reception sets the corresponding CAN*x*_IFLAG2 bit. If the corresponding CAN*x*_IMASK2 bit is set, an interrupt is generated. The interrupt flag must be cleared by writing it to ‘1’. Writing ‘0’ has no effect.

When the AEN bit in the CAN*x*_MCR is set (abort enabled), while the CAN*x*_IFLAG2 bit is set for a MB configured as Tx, the writing access done by CPU into the corresponding MB is blocked.

Offset: Base + 0x002C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	BUF 63I	BUF 62I	BUF 61I	BUF 60I	BUF 59I	BUF 58I	BUF 57I	BUF 56I	BUF 55I	BUF 54I	BUF 53I	BUF 52I	BUF 51I	BUF 50I	BUF 49I	BUF 48I
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	BUF 47I	BUF 46I	BUF 45I	BUF 44I	BUF 43I	BUF 42I	BUF 41I	BUF 40I	BUF 39I	BUF 38I	BUF 37I	BUF 36I	BUF 35I	BUF 34I	BUF 33I	BUF 32I
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-13. Interrupt Flag 2 Register (CAN*x*_IFLAG2)

Table 29-14. CAN*x*_IFLAG2 Field Descriptions

Field	Description
BUF*n*I	Message Buffer *n* Interrupt. Each bit represents the respective FlexCAN message buffer (MB63–MB32) interrupt. Write 1 to clear. 0 No such occurrence. 1 The corresponding buffer has successfully completed transmission or reception.

29.3.4.10 Interrupt Flags 1 Register (CAN*x*_IFLAG1)

This register defines the flags for 32 message buffer interrupts and FIFO interrupts. It contains one interrupt flag bit per buffer. Each successful transmission or reception sets the corresponding CAN*x*_IFLAG1 bit. If the corresponding CAN*x*_IMASK1 bit is set, an interrupt is generated. The interrupt flag must be cleared by writing it to ‘1’. Writing ‘0’ has no effect.

When the AEN bit in the CAN*x*_MCR register is set (Abort enabled), while the CAN*x*_IFLAG1 bit is set for a MB configured as Tx, the writing access done by CPU into the corresponding MB is blocked.

When the FEN bit in the CAN*x*_MCR register is set (FIFO enabled), the function of the eight least significant interrupt flags (BUF7I – BUF0I) is changed to support the FIFO operation. BUF7I, BUF6I and BUF5I indicate operating conditions of the FIFO, while BUF4I to BUF0I are not used.

Offset: Base + 0x0030 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	BUF 31I	BUF 30I	BUF 29I	BUF 28I	BUF 27I	BUF 26I	BUF 25I	BUF 24I	BUF 23I	BUF 22I	BUF 21I	BUF 20I	BUF 19I	BUF 18I	BUF 17I	BUF 16I
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	BUF 15I	BUF 14I	BUF 13I	BUF 12I	BUF 11I	BUF 10I	BUF 9I	BUF 8I	BUF 7I	BUF 6I	BUF 5I	BUF 4I	BUF 3I	BUF 2I	BUF 1I	BUF 0I
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-14. Interrupt Flag 1 Register (CAN*x*_IFLAG1)

Table 29-15. CAN*x*_IFLAG1 Field Descriptions

Field	Description
BUF31I–BUF8I	Message Buffer *n* Interrupt. Each bit represents the respective FlexCAN message buffer (MB31 to MB8) interrupt. Write 1 to clear. 0 No such occurrence. 1 The corresponding buffer has successfully completed transmission or reception.
BUF7I	Buffer MB7 Interrupt or “FIFO Overflow” If the FIFO is not enabled, this bit flags the interrupt for MB7. If the FIFO is enabled, this flag indicates an overflow condition in the FIFO (frame lost because FIFO is full). 0 No such occurrence. 1 MB7 completed transmission/reception or FIFO overflow.
BUF6I	Buffer MB6 Interrupt or “FIFO Warning” If the FIFO is not enabled, this bit flags the interrupt for MB6. If the FIFO is enabled, this flag indicates that 5 out of 6 buffers of the FIFO are already occupied (FIFO almost full). 0 No such occurrence. 1 MB6 completed transmission/reception or FIFO almost full.
BUF5I	Buffer MB5 Interrupt or “Frames Available in FIFO” If the FIFO is not enabled, this bit flags the interrupt for MB5. If the FIFO is enabled, this flag indicates that at least one frame is available to be read from the FIFO. 0 No such occurrence. 1 MB5 completed transmission/reception or frames available in the FIFO.
BUF4I–BUF0I	Buffer MB*n* Interrupt or “Reserved” If the FIFO is not enabled, these bits flag the interrupts for MB0 to MB4. If the FIFO is enabled, these flags are not used and must be considered as reserved locations. 0 No such occurrence. 1 Corresponding MB completed transmission/reception.

29.3.4.11 Rx Individual Mask Registers (CAN*x*_RXIMR0 – CAN*x*_RXIMR63)

These registers are used as acceptance masks for ID filtering in Rx MBs and the FIFO. If the FIFO is not enabled, one mask register is provided for each available message buffer, providing ID masking capability on a per message buffer basis. When the FIFO is enabled (FEN bit in CAN*x*_MCR is set), the first eight mask registers apply to the eight elements of the FIFO filter table (on a one-to-one correspondence), while the rest of the registers apply to the regular MBs, starting from MB8.

The individual Rx mask registers are implemented in RAM, so they are not affected by reset and must be explicitly initialized prior to any reception. Furthermore, they can only be accessed by the CPU while the module is in freeze mode. Out of freeze mode, write accesses are blocked and read accesses return all zeros. Furthermore, if the BCC bit in the register is negated, any read or write operation to these registers results in access error.

Offset: Base + 0880 - 0x0975 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	MI31	MI30	MI29	MI28	MI27	MI26	MI25	MI24	MI23	MI22	MI21	MI20	MI19	MI18	MI17	MI16
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	MI15	MI14	MI13	MI12	MI11	MI10	MI9	MI8	MI7	MI6	MI5	MI4	MI3	MI2	MI1	MI0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-15. Rx Individual Mask Registers (CAN$\bar{x}$_RXIMR0 – CAN*x*_RXIMR63)

Table 29-16. CAN*x*_RXIMR0 – CAN*x*_RXIMR63 Field Descriptions

Field	Description
MI31–M0	Mask Bits For normal Rx MBs, the mask bits affect the ID filter programmed on the MB. For the Rx FIFO, the mask bits affect all bits programmed in the filter table (ID, IDE, RTR). 0 The corresponding bit in the filter is “don’t care.” 1 The corresponding bit in the filter is checked against the one received.

29.4 Functional Description

The FlexCAN module is a CAN protocol engine with a very flexible mailbox system for transmitting and receiving CAN frames. The mailbox system is composed by a set of as many as 64 message buffers (MB) that store configuration and control data, time stamp, message ID and data (see Section 29.3.2, Message Buffer Structure). The memory corresponding to the first eight MBs can be configured to support a FIFO reception scheme with a powerful ID filtering mechanism, capable of checking incoming frames against a table of IDs (as many as eight extended IDs or 16 standard IDs or 32 8-bit ID slices), each one with its own individual mask register. Simultaneous reception through FIFO and mailbox is supported. For mailbox reception, a matching algorithm makes it possible to store received frames only into MBs that have the same ID programmed on its ID field. A masking scheme makes it possible to match the ID programmed on the MB with a range of IDs on received CAN frames. For transmission, an arbitration algorithm decides the prioritization of MBs to be transmitted based on the message ID (optionally augmented by 3 local priority bits) or the MB ordering.

Before proceeding with the functional description, an important concept must be explained. A message buffer is said to be “active” at a given time if it can participate in the matching and arbitration algorithms that are happening at that time. An Rx MB with a ‘0000’ code is inactive (refer to Table 29-4). Similarly, a Tx MB with a ‘1000’ or ‘1001’ code is also inactive (refer to Table 29-5). An MB not programmed with ‘0000’, ‘1000’ or ‘1001’ is temporarily deactivated (does not participate in the current arbitration or matching run) when the CPU writes to the C/S field of that MB (see Section 29.4.5.2, Message Buffer Deactivation).

29.4.1 Transmit Process

If the MB is active (transmission pending), write an ABORT code (‘1001’) to the code field of the control and status word to request an abortion of the transmission, then read back the code field and the IFLAG1/2 register to check if the transmission was aborted (see Section 29.4.5.1, Transmission Abort Mechanism). If backwards compatibility is desired (AEN in CAN*x*_MCR negated), just write ‘1000’ to the Code field to inactivate the MB but then the pending frame may be transmitted without notification (see Section 29.4.5.2, Message Buffer Deactivation).

- Write the ID word.
- Write the data bytes.
- Write the length, control and code fields of the control and status word to activate the MB.

Once the MB is activated in the fourth step, it participates into the arbitration process and eventually is transmitted according to its priority. At the end of the successful transmission, the value of the free-running timer is written into the time stamp field, the code field in the control and status word is updated, a status flag is set in the interrupt flag register and an interrupt is generated if allowed by the corresponding interrupt mask register bit. The new code field after transmission depends on the code that was used to activate the MB in step four (see Table 29-4 and Table 29-5 in Section 29.3.2, Message Buffer Structure). When the abort feature is enabled (AEN in CAN*x*_MCR is asserted), after the Interrupt Flag is asserted for a MB configured as transmit buffer, the MB is blocked, therefore the CPU is not able to update it until the interrupt flag be negated by CPU. It means that the CPU must clear the corresponding CAN*x*_IFLAG before starting to prepare this MB for a new transmission or reception.

29.4.2 Arbitration Process

This process selects which MB is transmitted next. All MBs programmed as transmit buffers are scanned to find the lowest ID[1] or the lowest MB number or the highest priority, depending on the LBUF and LPRIO_EN bits on the control register. The arbitration process is triggered in the following events:

- During the CRC field of the CAN frame
- During the error delimiter field of the CAN frame
- During intermission, if the winner MB defined in a previous arbitration was deactivated, or if there was no MB to transmit, but the CPU wrote to the C/S word of any MB after the previous arbitration finished
- When MBM is in idle or bus off state and the CPU writes to the C/S word of any MB
- Upon leaving freeze mode

When LBUF is asserted, the LPRIO_EN bit has no effect and the lowest number buffer is transmitted first. When LBUF and LPRIO_EN are both negated, the MB with the lowest ID is transmitted first but. If LBUF is negated and LPRIO_EN is asserted, the PRIO bits augment the ID used during the arbitration process. With this extended ID concept, arbitration is done based on the full 32-bit ID and the PRIO bits define which MB should be transmitted first, therefore MBs with PRIO = 000 have higher priority. If two or more MBs have the same priority, the regular ID determines the priority of transmission. If two or more MBs have the same priority (3 extra bits) and the same regular ID, the lowest MB is transmitted first.

1. If LBUF is negated, the arbitration considers not only the ID, but also the RTR and IDE bits placed inside the ID at the same positions they are transmitted in the CAN frame.

Once the highest priority MB is selected, it is transferred to a temporary storage space called serial message buffer (SMB), which has the same structure as a normal MB but is not user accessible. This operation is called “move-out” and after it is done, write access to the corresponding MB is blocked (if the AEN bit in CAN*x*_MCR is asserted). The write access is released in the following events:

- After the MB is transmitted
- FlexCAN enters in HALT or BUS OFF
- FlexCAN loses the bus arbitration or there is an error during the transmission

At the first opportunity window on the CAN bus, the message on the SMB is transmitted according to the CAN protocol rules. FlexCAN transmits as many as 8 data bytes, even if the data length code (DLC) value is bigger.

29.4.3 Receive Process

The CPU prepares a message buffer for frame reception by executing the following steps:

1. If the MB has a pending transmission, write an ABORT code (‘1001’) to the code field of the control and status word to request an abortion of the transmission, then read back the code field and the CAN*x*_IFLAG1/2 register to check if the transmission was aborted (see Section 29.4.5.1, Transmission Abort Mechanism). If backwards compatibility is desired (AEN in CAN*x*_MCR negated), just write ‘1000’ to the code field to inactivate the MB, but then the pending frame may be transmitted without notification (see Section 29.4.5.2, Message Buffer Deactivation). If the MB already programmed as a receiver, just write ‘0000’ to the code field of the control and status word to keep the MB inactive.
2. Write the ID word.
3. Write ‘0100’ to the code field of the control and status word to activate the MB.

After the MB is activated in the third step, it can receive frames that match the programmed ID. At the end of a successful reception, the MB is updated by the MBM as follows:

1. The value of the free-running timer is written into the time stamp field.
2. The received ID, data (8 bytes at most), and length fields are stored.
3. The code field in the control and status word is updated (see Table 29-4 and Table 29-5 in Section 29.3.2, Message Buffer Structure).
4. A status flag is set in the interrupt flag register and an interrupt is generated if allowed by the corresponding interrupt mask register bit.

Upon receiving the MB interrupt, the CPU should service the received frame using the following procedure:

1. Read the control and status word (mandatory – activates an internal lock for this buffer).
2. Read the ID field (optional – needed only if a mask was used).
3. Read the data field.
4. Read the free-running timer (optional – releases the internal lock).

Upon reading the control and status word, if the BUSY bit is set in the code field, then the CPU should defer the access to the MB until this bit is negated. Reading the free-running timer is not mandatory. If not

executed, the MB remains locked, unless the CPU reads the C/S word of another MB. Only a single MB is locked at a time. The only mandatory CPU read operation is the one on the control and status word to assure data coherency (see Section 29.4.5, Data Coherence).

The CPU should synchronize to frame reception by the status flag bit for the specific MB in one of the CAN*x*_IFLAG registers and not by the code field of that MB. Polling the code field does not work because after a frame is received and the CPU services the MB (by reading the C/S word followed by unlocking the MB), the code field does not return to EMPTY. It remains FULL, as explained in Table 29-4. If the CPU tries to work around this behavior by writing to the C/S word to force an EMPTY code after reading the MB, the MB is actually deactivated from any currently ongoing matching process. As a result, a newly received frame matching the ID of that MB may be lost. In summary: *never do polling by reading directly the C/S word of the MBs. Instead, read the CANx_IFLAG registers.*

The received ID field is always stored in the matching MB, thus the contents of the ID field in an MB may change if the match was due to masking. Note also that FlexCAN does receive frames transmitted by itself if there exists an Rx matching MB, provided the SRX_DIS bit in the CAN*x*_MCR is not asserted. If SRX_DIS is asserted, FlexCAN does not store frames transmitted by itself in any MB, even if it contains a matching MB, and no interrupt flag or interrupt signal is generated due to the frame reception.

To be able to receive CAN frames through the FIFO, the CPU must enable and configure the FIFO during freeze mode (see Section 29.4.6, Rx FIFO). Upon receiving the frames available interrupt from FIFO, the CPU should service the received frame using the following procedure:

1. Read the control and status word (optional – needed only if a mask was used for IDE and RTR bits).
2. Read the ID field (optional – needed only if a mask was used).
3. Read the data field.
4. Clear the frames available interrupt (mandatory – release the buffer and allow the CPU to read the next FIFO entry).

29.4.4 Matching Process

The matching process is an algorithm executed by the MBM that scans the MB memory looking for Rx MBs programmed with the same ID as the one received from the CAN bus. If the FIFO is enabled, the 8-entry ID table from FIFO is scanned first and then, if a match is not found within the FIFO table, the other MBs are scanned. In the event that the FIFO is full, the matching algorithm always looks for a matching MB outside the FIFO region.

When the frame is received, it is temporarily stored in a hidden auxiliary MB called serial message buffer (SMB). The matching process takes place during the CRC field of the received frame. If a matching ID is found in the FIFO table or in one of the regular MBs, the contents of the SMB are transferred to the FIFO or to the matched MB during the 6th bit of the end-of-frame field of the CAN protocol. This operation is called move-in. If any protocol error (CRC, ACK, etc.) is detected, than the move-in operation does not happen.

For the regular mailbox MBs, an MB is said to be free to receive a new frame if the following conditions are satisfied:

- The MB is not locked (see Section 29.4.5.3, Message Buffer Lock Mechanism)

- The code field is either EMPTY or else it is FULL or OVERRUN but the CPU has already serviced the MB (read the C/S word and then unlocked the MB)

If the first MB with a matching ID is not free to receive the new frame, then the matching algorithm keeps looking for another free MB until it finds one. If it cannot find one that is free, then it overwrites the last matching MB (unless it is locked) and sets the code field to OVERRUN (refer to Table 29-4 and Table 29-5). If the last matching MB is locked, then the new message remains in the SMB, waiting for the MB to be unlocked (see Section 29.4.5.3, Message Buffer Lock Mechanism).

Suppose, for example, that the FIFO is disabled and there are two MBs with the same ID, and FlexCAN starts receiving messages with that ID. Let us say that these MBs are the second and the fifth in the array. When the first message arrives, the matching algorithm finds the first match in MB number 2. The code of this MB is EMPTY, so the message is stored there. When the second message arrives, the matching algorithm finds MB number 2 again, but it is not free to receive, so it keeps looking and finds MB number 5 and stores the message there. If yet another message with the same ID arrives, the matching algorithm finds out that there are no matching MBs that are free to receive, so it decides to overwrite the last matched MB, which is number 5. In doing so, it sets the code field of the MB to indicate OVERRUN.

The ability to match the same ID in more than one MB can be exploited to implement a reception queue (in addition to the full featured FIFO) to allow more time for the CPU to service the MBs. By programming more than one MB with the same ID, received messages are queued into the MBs. The CPU can examine the time stamp field of the MBs to determine the order in which the messages arrived.

The matching algorithm described above can be changed to be the same one used in previous versions of the FlexCAN module. When the BCC bit in CAN*x*_MCR is negated, the matching algorithm stops at the first MB with a matching ID that it founds, whether this MB is free or not. As a result, the message queueing feature does not work if the BCC bit is negated.

Matching to a range of IDs is possible by using ID Acceptance Masks. FlexCAN supports individual masking per MB. Please refer to Section 29.3.4.11, Rx Individual Mask Registers (CANx_RXIMR0 – CANx_RXIMR63). During the matching algorithm, if a mask bit is asserted, then the corresponding ID bit is compared. If the mask bit is negated, the corresponding ID bit is “don’t care”. Please note that the individual mask registers are implemented in RAM, so they are not initialized out of reset. Also, they can only be programmed if the BCC bit is asserted and while the module is in freeze mode.

FlexCAN also supports an alternate masking scheme with only three mask registers (RGXMASK, CAN*x*_RX14MASK, and CAN*x*_RX15MASK) for backwards compatibility. This alternate masking scheme is enabled when the BCC bit in the CAN*x*_MCR Register is negated.

29.4.5 Data Coherence

To maintain data coherency and FlexCAN proper operation, the CPU must obey the rules described in Section 29.4.1, Transmit Process, and Section 29.4.3, Receive Process. Any form of CPU accessing an MB structure within FlexCAN other than those specified may cause FlexCAN to behave in an unpredictable way.

29.4.5.1 Transmission Abort Mechanism

The abort mechanism provides a safe way to request the abortion of a pending transmission. A feedback mechanism is provided to inform the CPU if the transmission was aborted or if the frame could not be aborted and was transmitted instead. To maintain backwards compatibility, the abort mechanism must be explicitly enabled by asserting the AEN bit in the CAN*x*_MCR.

In order to abort a transmission, the CPU must write a specific abort code (1001) to the code field of the control and status word. When the abort mechanism is enabled, the active MBs configured as transmission must be aborted first and then they may be updated. If the abort code is written to an MB that is currently being transmitted, or to an MB that was already loaded into the SMB for transmission, the write operation is blocked and the MB is not deactivated, but the abort request is captured and kept pending until one of the following conditions are satisfied:

- The module loses the bus arbitration
- There is an error during the transmission
- The module is put into freeze mode

If none of conditions above are reached, the MB is transmitted correctly, the interrupt flag is set in the CAN*x*_IFLAG register and an interrupt to the CPU is generated (if enabled). The abort request is automatically cleared when the interrupt flag is set. In the other hand, if one of the above conditions is reached, the frame is not transmitted, therefore the abort code is written into the code field, the interrupt flag is set in the CAN*x*_IFLAG and an interrupt is (optionally) generated to the CPU.

If the CPU writes the abort code before the transmission begins internally, then the write operation is not blocked, therefore the MB is updated and no interrupt flag is set. In this way the CPU needs to read the abort code to make sure the active MB was deactivated. Although the AEN bit is asserted and the CPU wrote the abort code, in this case the MB is deactivated and not aborted, because the transmission did not start yet. One MB is only aborted when the abort request is captured and kept pending until one of the previous conditions are satisfied.

The abort procedure can be summarized as follows:

- CPU writes 1001 into the code field of the C/S word
- CPU reads the CODE field and compares it to the value that was written
- If the CODE field that was read is different from the value that was written, the CPU must read the corresponding CAN*x*_IFLAG to check if the frame was transmitted or it is being currently transmitted. If the corresponding CAN*x*_IFLAG is set, the frame was transmitted. If the corresponding CAN*x*_IFLAG is reset, the CPU must wait for it to be set, and then the CPU must read the CODE field to check if the MB was aborted (CODE = 1001) or it was transmitted (CODE = 1000).

29.4.5.2 Message Buffer Deactivation

Deactivation is mechanism provided to maintain data coherence when the CPU writes to the control and status word of active MBs out of freeze mode. Any CPU write access to the control and status word of an MB causes that MB to be excluded from the transmit or receive processes during the current matching or arbitration round. The deactivation is temporary, affecting only for the current match/arbitration round.

The purpose of deactivation is data coherency. The match/arbitration process scans the MBs to decide which MB to transmit or receive. If the CPU updates the MB in the middle of a match or arbitration process, the data of that MB may no longer be coherent, therefore deactivation of that MB is done.

Even with the coherence mechanism described above, writing to the control and status word of active MBs when not in freeze mode may produce undesirable results. Examples are:

- Matching and arbitration are one-pass processes. If MBs are deactivated after they are scanned, no re-evaluation is done to determine a new match/winner. If an Rx MB with a matching ID is deactivated during the matching process after it was scanned, then this MB is marked as invalid to receive the frame, and FlexCAN keeps looking for another matching MB within the ones it has not scanned yet. If it cannot find one, then the message is lost. Suppose, for example, that two MBs have a matching ID to a received frame, and the user deactivated the first matching MB after FlexCAN has scanned the second. The received frame is lost even if the second matching MB was free to receive.
- If a Tx MB containing the lowest ID is deactivated after FlexCAN has scanned it, then FlexCAN looks for another winner within the MBs that it has not scanned yet. Therefore, it may transmit an MB with ID that may not be the lowest at the time because a lower ID might be present in one of the MBs that it had already scanned before the deactivation.
- There is a point in time until which the deactivation of a Tx MB causes it not to be transmitted (end of move-out). After this point, it is transmitted but no interrupt is issued and the code field is not updated. In order to avoid this situation, the abort procedures described in Section 29.4.5.1, Transmission Abort Mechanism, should be used.

29.4.5.3 Message Buffer Lock Mechanism

Besides MB deactivation, FlexCAN has another data coherence mechanism for the receive process. When the CPU reads the Control and Status word of an “active not empty” Rx MB, FlexCAN assumes that the CPU wants to read the whole MB in an atomic operation, and thus it sets an internal lock flag for that MB. The lock is released when the CPU reads the free-running timer (global unlock operation), or when it reads the Control and Status word of another MB. The MB locking is done to prevent a new frame to be written into the MB while the CPU is reading it.

NOTE

The locking mechanism only applies to Rx MBs which have a code different than INACTIVE (‘0000’) or EMPTY[1] (‘0100’). Also, Tx MBs cannot be locked.

Suppose, for example, that the FIFO is disabled and the second and the fifth MBs of the array are programmed with the same ID, and FlexCAN has already received and stored messages into these two MBs. Suppose now that the CPU decides to read MB number 5 and at the same time another message with the same ID is arriving. When the CPU reads the control and status word of MB number 5, this MB is locked. The new message arrives and the matching algorithm finds out that there are no free to receive MBs, so it decides to override MB number 5. However, this MB is locked, so the new message cannot be written there. It remains in the SMB waiting for the MB to be unlocked, and not written to the MB until

1. In previous FlexCAN versions, reading the C/S word locked the MB even if it was EMPTY. This behavior is honored when the BCC bit is negated.

then. If the MB is not unlocked in time and yet another new message with the same ID arrives, then the new message overwrites the one on the SMB and there is no indication of lost messages either in the code field of the MB or in the error and status register.

While the message is being moved-in from the SMB to the MB, the BUSY bit on the code field is asserted. If the CPU reads the control and status word and finds out that the BUSY bit is set, it should defer accessing the MB until the BUSY bit is negated.

NOTE

If the BUSY bit is asserted or if the MB is empty, then reading the control and status word does not lock the MB.

Deactivation takes precedence over locking. If the CPU deactivates a locked Rx MB, then its lock status is negated and the MB is marked as invalid for the current matching round. Any pending message on the SMB is not transferred anymore to the MB.

29.4.6 Rx FIFO

The receive-only FIFO is enabled by asserting the FEN bit in the CAN*x*_MCR. The reset value of this bit is zero to maintain software backwards compatibility with previous versions of the module that did not have the FIFO feature. When the FIFO is enabled, the memory region normally occupied by the first 8 MBs (0x80-0xFF) is now reserved for use of the FIFO engine (see Section 29.3.3, Rx FIFO Structure). Management of read and write pointers is done internally by the FIFO engine. The CPU can read the received frames sequentially, in the order they were received, by repeatedly accessing a Message Buffer structure at the beginning of the memory.

The FIFO can store as many as six frames pending service by the CPU. An interrupt is sent to the CPU when new frames are available in the FIFO. Upon receiving the interrupt, the CPU must read the frame (accessing an MB in the 0x80 address) and then clear the interrupt. The act of clearing the interrupt triggers the FIFO engine to replace the MB in 0x80 with the next frame in the queue, and then issue another interrupt to the CPU. If the FIFO is full and more frames continue to be received, an OVERFLOW interrupt is issued to the CPU and subsequent frames are not accepted until the CPU creates space in the FIFO by reading one or more frames. A warning interrupt is also generated when 4 frames are accumulated in the FIFO.

A powerful filtering scheme is provided to accept only frames intended for the target application, thus reducing the interrupt servicing work load. The filtering criteria is specified by programming a table of 8 32-bit registers that can be configured to one of the following formats (see also Section 29.3.3, Rx FIFO Structure):

- Format A: 8 extended or standard IDs (including IDE and RTR)
- Format B: 16 standard IDs or 16 extended 14-bit ID slices (including IDE and RTR)
- Format C: 32 standard or extended 8-bit ID slices

NOTE

A chosen format is applied to all eight registers of the filter table. It is not possible to mix formats within the table.

The eight elements of the filter table are individually affected by the first eight individual mask registers (CAN*x*_RXIMR0 – CAN*x*_RXIMR7), allowing very powerful filtering criteria to be defined. The rest of the RXIMR, starting from RXIM8, continue to affect the regular MBs, starting from MB8. If the BCC bit is negated, then the FIFO filter table is affected by the legacy mask registers as follows: element 6 is affected by CAN*x*_RX14MASK, element 7 is affected by CAN*x*_RX15MASK and the other elements (0 to 5) are affected by CAN*x*_RXGMASK.

29.4.7 CAN Protocol Related Features

29.4.7.1 Remote Frames

A remote frame is a special kind of frame. The user can program a MB to be a request remote frame by writing the MB as transmit with the RTR bit set to 1. After the remote request frame is transmitted successfully, the MB becomes a receive message buffer, with the same ID as before.

When a remote request frame is received by FlexCAN, its ID is compared to the IDs of the transmit message buffers with the CODE field ‘1010’. If there is a matching ID, then this MB frame is transmitted. Note that if the matching MB has the RTR bit set, then FlexCAN transmits a remote frame as a response.

A received remote request frame is not stored in a receive buffer. It is only used to trigger a transmission of a frame in response. The mask registers are not used in remote frame matching, and all ID bits (except RTR) of the incoming received frame should match.

In the case that a remote request frame was received and matched a MB, this message buffer immediately enters the internal arbitration process, but is considered as normal Tx MB, with no higher priority. The data length of this frame is independent of the DLC field in the remote frame that initiated its transmission.

If the Rx FIFO is enabled (bit FEN set in CAN*x*_MCR), FlexCAN does not generate an automatic response for remote request frames that match the FIFO filtering criteria. If the remote frame matches one of the target IDs, it is stored in the FIFO and presented to the CPU. Note that for filtering formats A and B, it is possible to select whether remote frames are accepted or not. For format C, remote frames are always accepted (if they match the ID).

29.4.7.2 Overload Frames

FlexCAN transmits overload frames due to detection of these conditions on CAN bus:

- Detection of a dominant bit in the first/second bit of intermission
- Detection of a dominant bit at the 7th bit (last) of end of frame field (Rx frames)
- Detection of a dominant bit at the 8th bit (last) of error frame delimiter or overload frame delimiter

29.4.7.3 Time Stamp

The value of the free-running timer is sampled at the beginning of the identifier field on the CAN bus, and is stored at the end of move in the TIME STAMP field, providing network behavior with respect to time.

Note that the free-running timer can be reset on a specific frame reception, enabling network time synchronization. Refer to TSYN description in Section 29.3.4.2, Control Register (CANx_CTRL).

29.4.7.4 Protocol Timing

The clock source to the CAN protocol interface (CPI) can be either the system clock or a direct feed from the oscillator pin EXTAL. The clock source is selected by the CLK_SRC bit in the CAN*x*_CTRL. The clock is fed to the prescaler to generate the serial clock (SCK).

The FlexCAN module supports a variety of means to setup bit timing parameters that are required by the CAN protocol. The CAN*x*_CTRL has various fields used to control bit timing parameters: PRESDIV, PROPSEG, PSEG1, PSEG2 and RJW. See Section 29.3.4.2, Control Register (CANx_CTRL).

The PRESDIV field controls a prescaler that generates the serial clock (SCK), whose period defines the ‘time quantum’ used to compose the CAN waveform. A time quantum is the atomic unit of time handled by FlexCAN.

$$f_{Tq} = \frac{f_{CANCLK}}{\text{Prescaler Value}}$$

A bit time is subdivided into three segments[1] (reference Figure 29-16 and Table 29-17):

- SYNCSEG: This segment has a fixed length of one time quantum. Signal edges are expected to happen within this section.
- Time segment 1: This segment includes the propagation segment and the phase segment 1 of the CAN standard. It can be programmed by setting the PROPSEG and the PSEG1 fields of the CAN*x*_CTRL register so that their sum (plus 2) is in the range of 4 to 16 time quanta.
- Time segment 2: This segment represents the phase segment 2 of the CAN standard. It can be programmed by setting the PSEG2 field of the CAN*x*_CTRL register (plus 1) to be 2 to 8 time quanta long.

$$\text{Bit Rate} = \frac{f_{Tq}}{(\text{Number of Time Quanta})}$$

1. For further explanation of the underlying concepts please refer to ISO/DIS 11519–1, Section 10.3. Reference also the Bosch CAN 2.0A/B protocol specification dated September 1991 for bit timing.

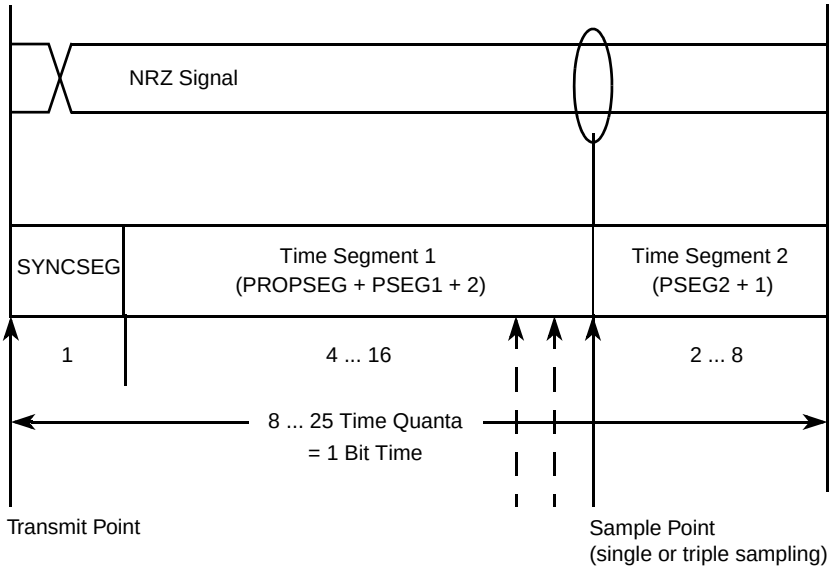


Figure 29-16. Segments within the Bit Time

Table 29-17. Time Segment Syntax

Syntax	Description
SYNCSEG	System expects transitions to occur on the bus during this period.
Transmit Point	A node in transmit mode transfers a new value to the CAN bus at this point.
Sample Point	A node in receive mode samples the bus at this point. If the three samples per bit option is selected, then this point marks the position of the third sample.

Table 29-18 gives an overview of the CAN compliant segment settings and the related parameter values.

NOTE

It is the user's responsibility to ensure the bit time settings are in compliance with the CAN standard. For bit time calculations, use an information processing time (IPT) of 2, which is the value implemented in the FlexCAN module.

Table 29-18. CAN Standard Compliant Bit Time Segment Settings

Time Segment 1	Time Segment 2	Resynchronization Jump Width
5 .. 10	2	1 .. 2
4 .. 11	3	1 .. 3
5 .. 12	4	1 .. 4
6 .. 13	5	1 .. 4
7 .. 14	6	1 .. 4

Table 29-18. CAN Standard Compliant Bit Time Segment Settings (continued)

Time Segment 1	Time Segment 2	Resynchronization Jump Width
8 .. 15	7	1 .. 4
9 .. 16	8	1 .. 4

29.4.7.5 Arbitration and Matching Timing

During normal transmission or reception of frames, the arbitration, match, move in and move out processes are executed during certain time windows inside the CAN frame, as shown in Figure 29-17. When doing matching and arbitration, FlexCAN needs to scan the whole message buffer memory during the available time slot. In order to have sufficient time to do that, the following restrictions must be observed:

- A valid CAN bit timing must be programmed, as indicated in Figure 29-17.
- The system clock frequency cannot be smaller than the oscillator clock frequency, i.e.,the PLL cannot be programmed to divide down the oscillator clock.
- There must be a minimum ratio of 16 between the system clock frequency and the CAN bit rate.

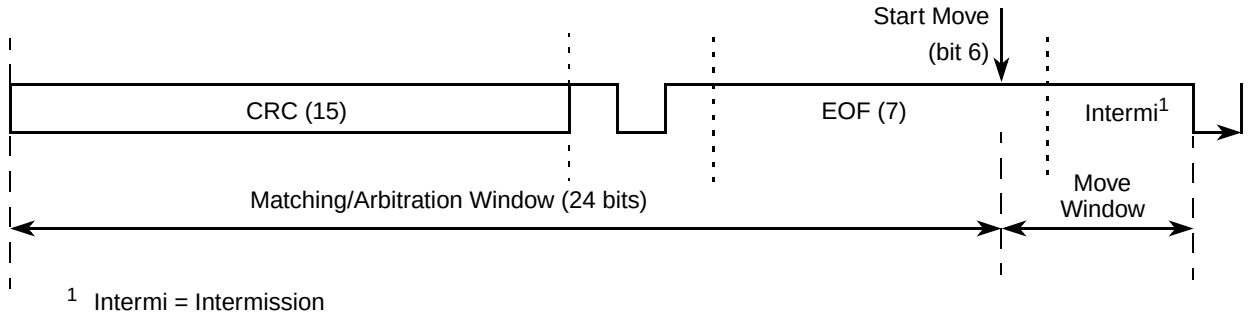


Figure 29-17. Arbitration, Match and Move Time Windows

29.4.8 Modes of Operation Details

29.4.8.1 Freeze Mode

This mode is entered by asserting the HALT bit in the CAN*x*_MCR or when the MCU is put into debug mode. In both cases it is also necessary that the FRZ bit is asserted in the CAN*x*_MCR. When freeze mode is requested during transmission or reception, FlexCAN does the following:

- Waits to be in either intermission, passive error, bus off or idle state
- Waits for all internal activities like move in or move out to finish
- Ignores the Rx input pin and drives the Tx pin as recessive
- Stops the prescaler, thus halting all CAN protocol activities
- Grants write access to the CAN*x*_ECR, which is read-only in other modes
- Sets the NOT_RDY and FRZ_ACK bits in CAN*x*_MCR

After requesting freeze mode, the user must wait for the FRZ_ACK bit to be asserted in CAN*x*_MCR before executing any other action, otherwise FlexCAN can operate in an unpredictable way. In freeze mode, all memory mapped registers are accessible.

Exiting freeze mode is done in one of these ways:

- CPU negates the FRZ bit in the CAN*x*_MCR
- The MCU exits debug mode and/or the HALT bit is negated

After it is out of freeze mode, FlexCAN tries to resynchronize to the CAN bus by waiting for 11 consecutive recessive bits.

29.4.8.2 Module Disabled Mode

This low-power mode is entered when the CAN*x*_MCR[MDIS] bit is asserted. If the module is disabled during freeze mode, it shuts down the clocks to the CPI and MBM sub-modules, sets the CAN*x*_MCR[LPM_ACK] bit, and negates the CAN*x*_MCR[FRZ_ACK] bit. If the module is disabled during transmission or reception, FlexCAN does the following:

- Waits to be in either idle or bus off state, or else waits for the third bit of intermission and then checks it to be recessive
- Waits for all internal activities like move in or move out to finish
- Ignores its Rx input pin and drives its Tx pin as recessive
- Shuts down the clocks to the CPI and MBM sub-modules
- Sets the NOT_RDY and LPM_ACK bits in CAN*x*_MCR

The bus interface unit continues to operate, enabling the CPU to access memory mapped registers except the free-running timer, the CAN*x*_ECR and the message buffers, which cannot be accessed when the module is disabled. Exiting from this mode is done by negating the CAN*x*_MCR[MDIS] bit, which resumes the clocks and negates the CAN*x*_MCR[LPM_ACK] bit.

29.4.9 Interrupts

The FlexCAN module interrupts are ORed together at the chip level as described in Chapter 10, Interrupts and Interrupt Controller (INTC).

There is an interrupt source for each MB from MB0 to MB15. There is no distinction between Tx and Rx interrupts for a particular buffer, under the assumption that the buffer is initialized for either transmission or reception. Each of the buffers has assigned a flag bit in the CAN*x*_IFLAG2 or CAN*x*_IFLAG1 registers. The bit is set when the corresponding buffer completes a successful transmission/reception and is cleared when the CPU writes it to 1.

A combined interrupt for each of two MB groups, MB16–MB31 and MB32–MB63, is also generated by an OR of all the interrupt sources from the associated MBs. This interrupt gets generated when any of the MBs generates an interrupt. In this case the CPU must read the CAN*x*_IFLAG2 and CAN*x*_IFLAG1 registers to determine which MB caused the interrupt.

The other two interrupt sources (bus off/transmit warning/receive warning and error) generate interrupts like the MB interrupt sources, and can be read from CAN*x*_ESR. The bus off/transmit warning/receive warning and error interrupt mask bits are located in the CAN*x*_CTRL.

29.4.10 Bus Interface

The CPU access to FlexCAN registers are subject to the following rules:

- Read and write access to unimplemented or reserved address space results in access error. Any access to unimplemented MB Rx individual mask register locations results in access error. Any access to the Rx individual mask register space when the BCC bit in CAN*x*_MCR is negated results in access error.
- For a FlexCAN configuration that uses less than the total number of MBs and MAXMB is set accordingly, the remaining MB and Rx mask register spaces can be used as general-purpose RAM space. Note that the Rx individual mask registers can only be accessed in freeze mode, and this is still true for unused space within this memory. Note also that reserved words within RAM cannot be used. As an example, suppose FlexCAN is configured with 64 MBs and MAXMB is programmed with zero. The maximum number of MBs in this case becomes one. The MB memory starts at 0x0060, but the space from 0x0060 to 0x007F is reserved (for SMB usage), and the space from 0x0080 to 0x008F is used by the one MB. This leaves us with the available space from 0x0090 to 0x047F. The available memory in the mask registers space would be from 0x0884 to 0x097F. Byte, word, and long word accesses are allowed to the unused MB space.

NOTE

Unused MB space must not be used as general purpose RAM while FlexCAN is transmitting and receiving CAN frames.

29.5 Initialization and Application Information

This section provides instructions for initializing the FlexCAN module.

29.5.1 FlexCAN Initialization Sequence

The FlexCAN module can be reset in three ways:

- MCU-level hard reset, which resets all memory-mapped registers asynchronously
- MCU-level soft reset, which resets some of the memory-mapped registers synchronously (refer to Table 29-7 to see what registers are affected by soft reset)
- SOFT_RST bit in CAN*x*_MCR, which has the same effect as the MCU level soft reset

Soft reset is synchronous and has to follow an internal request/acknowledge procedure across clock domains. Therefore, it may take some time to fully propagate its effects. The SOFT_RST bit remains asserted while soft reset is pending, so software can poll this bit to know when the reset has completed.

After the module is enabled (CAN*x*_MCR[MDIS] bit negated), FlexCAN must be put into freeze mode before doing any configuration. In freeze mode, FlexCAN is un-synchronized to the CAN bus, the HALT and FRZ bits in CAN*x*_MCR are set, the internal state machines are disabled and the FRZ_ACK and

NOT_RDY bits in the CAN*x*_MCR are set. The CNTX pin is in recessive state and FlexCAN does not initiate frame transmission nor receives any frames from the CAN bus. Note that the message buffer contents are not affected by reset, so they are not automatically initialized.

For any configuration change/initialization, it is required that FlexCAN is put into freeze mode (see Section 29.4.8.1, Freeze Mode). The following is a generic initialization sequence applicable for the FlexCAN module:

- Initialize the CAN*x*_MCR
  - Enable the individual filtering per MB and reception queue features by setting the BCC bit
  - Enable the warning interrupts by setting the WRN_EN bit
  - If required, disable frame self reception by setting the SRX_DIS bit
  - Enable the FIFO by setting the FEN bit
  - Enable the abort mechanism by setting the AEN bit
  - Enable the local priority feature by setting the LPRIO_EN bit
- Initialize CAN*x*_CTRL.
  - Determine bit timing parameters: PROPSEG, PSEG1, PSEG2, RJW.
  - Determine the bit rate by programming the PRESDIV field.
  - Determine internal arbitration mode (LBUF bit).
- Initialize message buffers.
  - The control and status word of all message buffers must be initialized
  - If FIFO was enabled, the 8-entry ID table must be initialized
  - Other entries in each message buffer should be initialized as required
- Initialize the Rx individual mask registers
- Set required interrupt mask bits in the IMASK registers (for all MB interrupts), in CAN*x*_CTRL (for bus off and error interrupts) and in CAN*x*_MCR for wake-up interrupt
- Negate the HALT bit in CAN*x*_MCR

Starting with this last event, FlexCAN attempts to synchronize with the CAN bus.

Chapter 30
Deserial – Serial Peripheral Interface (DSPI)

30.1 Introduction

The deserial serial peripheral interface (DSPI) block provides a synchronous serial interface for communication between the PXN20 and external devices. The DSPI supports pin-count reduction through serialization and deserialization of eMIOS channels and memory-mapped registers. The channels and register content are transmitted using a SPI-like protocol. There are four identical DSPI blocks: DSPI_A, DSPI_B, DSPI_C, and DSPI_D.

The DSPIs have three configurations:

- Serial peripheral interface (SPI) configuration where the DSPI operates as a basic SPI or as a queued SPI through the use of internal FIFOs.
- Deserial serial interface (DSI) configuration where the DSPI serializes the parallel input signals and deserializes received data by placing it on the parallel output signals.
- Combined serial interface (CSI) configuration where the DSPI operates in both SPI and DSI configurations interleaving DSI frames with SPI frames, giving priority to SPI frames.

For queued operations, the SPI queues reside in system memory external to the DSPI. Data transfers between the memory and the DSPI FIFOs are accomplished through the use of the eDMA controller or through host software. Figure 30-1 shows a DSPI with external queues in system RAM.

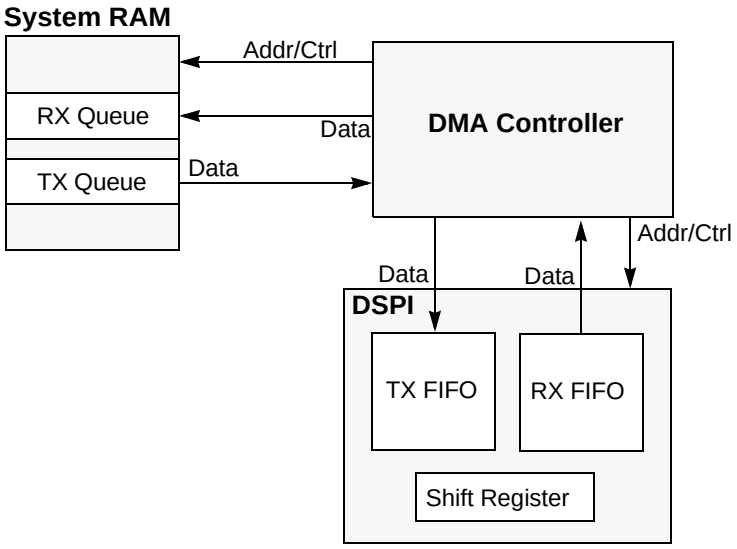


Figure 30-1. DSPI with Queues and DMA

30.1.1 Block Diagram

Figure 30-2 is a simplified block diagram of the DSPI that illustrates the functionality and interdependence of major blocks.

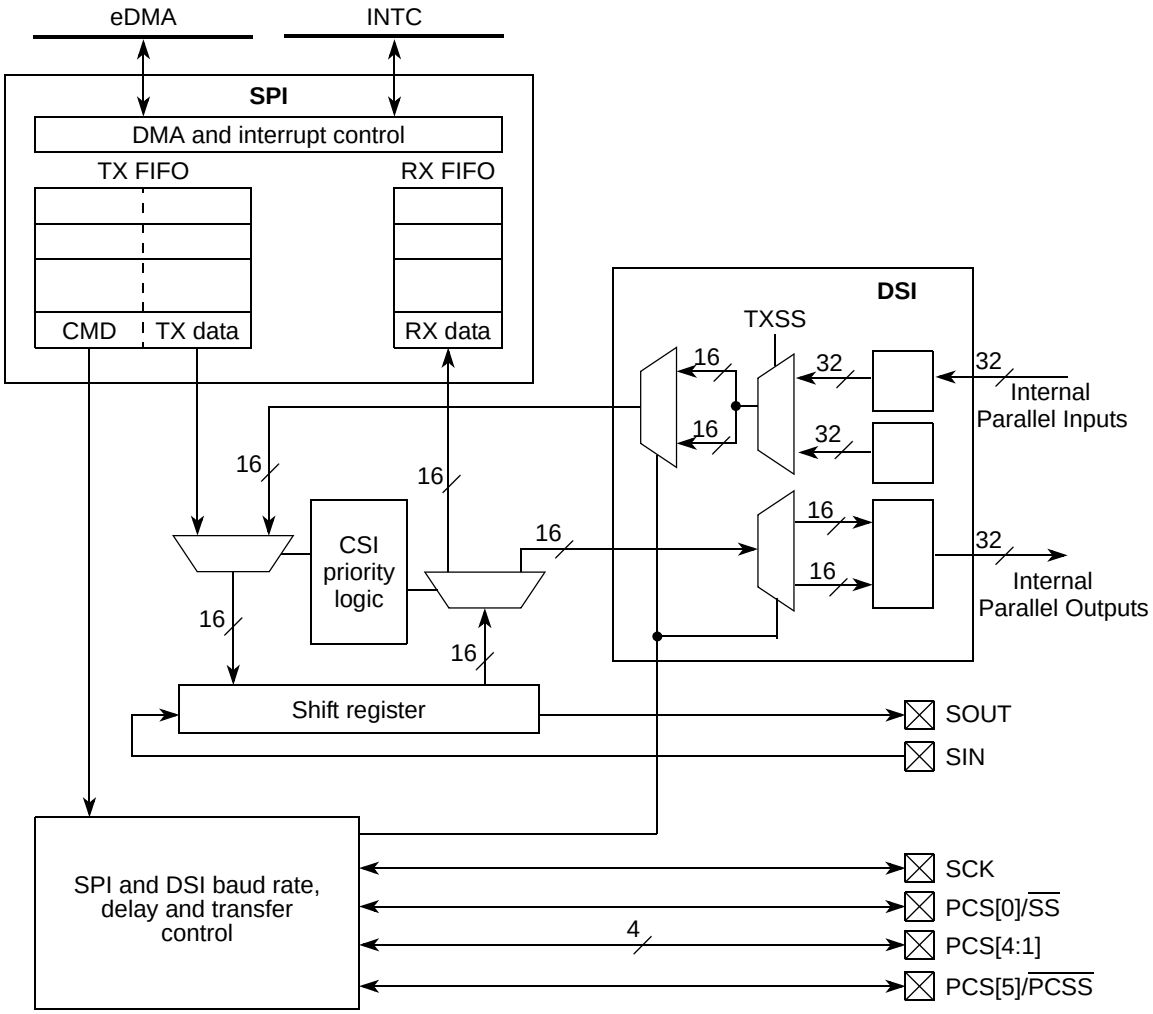


Figure 30-2. DSPI Block Diagram

30.1.2 Features

The DSPI supports these SPI features:

- Full-duplex, three-wire synchronous transfers
- Master and slave mode
- Buffered transmit operation using the TX FIFO with depth of 4 entries
- Buffered receive operation using the RX FIFO with depth of 4 entries
- TX and RX FIFOs can be disabled individually for low-latency updates to SPI queues
- Visibility into TX and RX FIFOs for ease of debugging
- Programmable transfer attributes on a per-frame basis:

  - Parameterized number of transfer attribute registers (from two to eight)
  - Serial clock with programmable polarity and phase
  - Various programmable delays
  - Programmable serial frame size of 4 to 32 bits, expandable by software control
  - Continuously held chip select capability
- Six peripheral chip selects, expandable to 32 with external demultiplexer
- Deglitching support for as many as 128 peripheral chip select with external demultiplexer
- DMA support for adding entries to TX FIFO and removing entries from RX FIFO:
  - TX FIFO is not full (TFFF)
  - RX FIFO is not empty (RFDF)
- Six Interrupt conditions:
  - End of queue reached (EOQF)
  - TX FIFO is not full (TFFF)
  - Transfer of current frame complete (TCF)
  - Attempt to transmit with an empty Transmit FIFO (TFUF)
  - RX FIFO is not empty (RFDF)
  - Frame received while Receive FIFO is full (RFOF)
- Modified SPI transfer formats for communication with slower peripheral devices
- Module disable mode supported via MDIS bits in the DSPI block
- Halt mode supported via HLT bits in the SIU block

The DSPI also supports pin reduction through serialization and deserialization.

- Two sources of serialized data:
  - DSPI memory-mapped register
  - Parallel Input signals
- Deserialized data is provided as Parallel Output signals and as bits in a memory-mapped register
- Transfer initiation conditions:
  - Continuous
  - Change in data
- Pin serialization/deserialization with interleaved SPI frames for control and diagnostics
- Continuous serial communications clock
- Enhanced DSI logic to implement a 32-bit Timed Serial Bus (TSB) configuration, supporting the Micro Second Bus downstream frame format.

30.1.3 DSPI Configurations

The DSPI block has three distinct serial transmission configurations; SPI, DSI and CSI.

30.1.3.1 SPI Configuration

The SPI configuration allows the DSPI to send and receive serial data. This configuration allows the DSPI to operate as a basic SPI block with the FIFOs providing support for external queue operation. Data to be transmitted and data received reside in separate FIFOs. The FIFOs can be popped and pushed by host software or by a DMA controller.

30.1.3.2 DSI Configuration

In the DSI configuration, the DSPI serializes as many as 16 parallel input signals or register bits. The DSPI also deserializes the received data to parallel output signals or to a memory-mapped register. The data is transferred using a SPI-like protocol.

Specifically in the TSB configuration, detailed on Section 30.4.10, Timed Serial Bus (TSB), the DSPI serializes from 4 to 32 Parallel Input signals or register bits. The TSB downstream frame used to communicate with a single slave is shown in Figure 30-39.

30.1.3.3 CSI Configuration

The CSI configuration is a combination of the SPI and DSI configurations. In this configuration, the DSPI interleaves DSI data with SPI data. Interleaving is done on the frame boundaries. In this configuration, SPI data transmission has higher priority than DSI data transmission.

30.1.4 Modes of Operation

The DSPI has five modes of operation that can be divided into two categories; block-specific modes such as master, slave, and module disable modes; and MCU-specific modes like external halt and debug modes.

The block-specific modes are entered by host software writing to a register. The MCU-specific modes are controlled by signals external to the DSPI. The MCU-specific modes are modes that the entire MCU may enter, in parallel to the DSPI being in one of its block-specific modes.

30.1.4.1 Master Mode

Master mode allows the DSPI to initiate and control serial communication. In this mode, the SCK signal and the PCS[*x*] signals are controlled by the DSPI and configured as outputs.

30.1.4.2 Slave Mode

Slave mode allows the DSPI to communicate with SPI/DSI bus masters. In this mode the DSPI responds to externally controlled serial transfers. The DSPI cannot control serial transfers in slave mode. In this mode, the SCK signal and the PCS[0]/$\overline{\text{SS}}$ signal are configured as inputs and provided by a bus master.

30.1.4.3 Module Disable Mode

Module disable mode is used for MCU power management. The clock to the non-memory mapped logic in the DSPI can be stopped while in the module disable mode. Logic external to the DSPI is needed to fully implement the module disable mode.

30.1.4.4 Halt Mode

Halt mode is used for MCU power management and controlled by the individual HLT bits in the SIU_HLT0 register. When a request is made to enter halt mode (assert HLT bit), the DSPI block acknowledges the request and completes the transfer in progress. When the DSPI reaches the frame boundary it signals that the system clocks to the DSPI block may be shut off.

30.1.4.5 Debug Mode

Debug mode is used for system development and debugging. If the device enters debug mode while the FRZ bit in the DSPI_MCR is set, the DSPI stops all serial transfers. If the device enters debug mode while the FRZ bit is negated, the DSPI behavior is unaffected and remains dictated by the block-specific mode and configuration of the DSPI.

30.2 External Signal Description

The DSPI supports the following external signals:

Table 30-1. External Signals

Name	I/O Type	Function	
		Master Mode	Slave Mode
PCS[0] / $\overline{SS}$	Output / Input	Peripheral Chip Select 0	Slave Select
PCS[1] – PCS[4]	Output	Peripheral Chip Select 1–4	unused
PCS[5] / $\overline{PCSS}$	Output	Peripheral Chip Select 5 / Peripheral Chip Select Strobe	unused
SIN	Input	Serial Data In	
SOUT	Output	Serial Data Out	
SCK	Output / Input	Serial Clock (output)	Serial Clock (input)

Refer to Table 3-1 and Section 3.4, Detailed Signal Description, for detailed signal descriptions.

30.3 Memory Map and Registers

This section provides a detailed description of all DSPI registers.

30.3.1 Module Memory Map

The DSPI memory map is shown in Table 30-2 (the memory map is the same for each individual DSPI module). The address of each register is given as an offset to the DSPI base address. Registers are listed in address order, identified by complete name and mnemonic, and list the type of accesses allowed.

Table 30-2. DSPI Memory Map

Offset from DSPI_BASE DSPI_A = 0xFFF9_0000 DSPI_B = 0xFFF9_4000 DSPI_C = 0xC3F9_0000 DSPI_D = 0xC3F9_4000	Register	Access	Reset Value	Section/Page
0x0000	DSPI_MCR—DSPI module configuration register	R/W	0x0000_4001	30.3.2.1/30-7
0x0004	Reserved			
0x0008	DSPI_TCR—DSPI transfer count register	R/W	0x0000_0000	30.3.2.2/30-9
0x000C	DSPI_CTAR0—DSPI clock and transfer attributes register 0	R/W	0x7800_0000	30.3.2.3/30-10
0x0010	DSPI_CTAR1—DSPI clock and transfer attributes register 1	R/W	0x7800_0000	30.3.2.3/30-10
0x0014	DSPI_CTAR2—DSPI clock and transfer attributes register 2	R/W	0x7800_0000	30.3.2.3/30-10
0x0018	DSPI_CTAR3—DSPI clock and transfer attributes register 3	R/W	0x7800_0000	30.3.2.3/30-10
0x001C	DSPI_CTAR4—DSPI clock and transfer attributes register 4	R/W	0x7800_0000	30.3.2.3/30-10
0x0020	DSPI_CTAR5—DSPI clock and transfer attributes register 5	R/W	0x7800_0000	30.3.2.3/30-10
0x0024	DSPI_CTAR6—DSPI clock and transfer attributes register 6	R/W	0x7800_0000	30.3.2.3/30-10
0x0028	DSPI_CTAR7—DSPI clock and transfer attributes register 7	R/W	0x7800_0000	30.3.2.3/30-10
0x002C	DSPI_SR—DSPI status register	R	0x0000_0000	30.3.2.4/30-16
0x0030	DSPI_RSER—DSPI DMA/interrupt request select and enable register	R/W	0x0000_0000	30.3.2.5/30-18
FIFO Registers				
0x0034	DSPI_PUSHR—DSPI push TX FIFO register	R/W	0x0000_0000	30.3.2.6/30-19
0x0038	DSPI_POPR—DSPI pop RX FIFO register	R	0x0000_0000	30.3.2.7/30-21
0x003C	DSPI_TXFR0—DSPI transmit FIFO register 0	R	0x0000_0000	30.3.2.8/30-21
0x0040	DSPI_TXFR1—DSPI transmit FIFO register 1	R	0x0000_0000	30.3.2.8/30-21
0x0044	DSPI_TXFR2—DSPI transmit FIFO register 2	R	0x0000_0000	30.3.2.8/30-21
0x0048	DSPI_TXFR3—DSPI transmit FIFO register 3	R	0x0000_0000	30.3.2.8/30-21
0x004C–0x0078	Reserved			
0x007C	DSPI_RXFR0—DSPI receive FIFO register 0	R	0x0000_0000	30.3.2.9/30-22
0x0080	DSPI_RXFR1—DSPI receive FIFO register 1	R	0x0000_0000	30.3.2.9/30-22
0x0084	DSPI_RXFR2—DSPI receive FIFO register 2	R	0x0000_0000	30.3.2.9/30-22
0x0088	DSPI_RXFR3—DSPI receive FIFO register 3	R	0x0000_0000	30.3.2.9/30-22
0x008C–0x00B8	Reserved			
DSI Registers				
0x00BC	DSPI_DSICR—DSPI DSI configuration register	R/W	0x0000_0000	30.3.2.10/30-23
0x00C0	DSPI_SDR—DSPI DSI serialization data register	R	0x0000_0000	30.3.2.11/30-24
0x00C4	DSPI_ASDR—DSPI DSI alternate serialization data register	R/W	0x0000_0000	30.3.2.12/30-25
0x00C8	DSPI_COMPR—DSPI DSI transmit comparison register	R	0x0000_0000	30.3.2.13/30-26
0x00CC	DSPI_DDR—DSPI DSI deserialization data register	R	0x0000_0000	30.3.2.14/30-26

Table 30-2. DSPI Memory Map (continued)

Offset from DSPI_BASE DSPI_A = 0xFFF9_0000 DSPI_B = 0xFFF9_4000 DSPI_C = 0xC3F9_0000 DSPI_D = 0xC3F9_4000	Register	Access	Reset Value	Section/Page
0x00D0	DSPI_DSICR1—DSPI DSI TSB configuration register 1	R/W	0x0000_0000	30.3.2.15/30-27
0x00D4–0x3FFF	Reserved			

30.3.2 Register Descriptions

This section lists the DSPI registers in address order and describes the registers and their bit fields.

30.3.2.1 DSPI Module Configuration Register (DSPI_MCR)

The DSPI_MCR contains bits that configure various attributes associated with DSPI operation. The HALT and MDIS bits can be changed at any time but only takes effect on the next frame boundary. Only the HALT and MDIS bits in the DSPI_MCR may be changed while the DSPI is in the Running state.

Offset: DSPI_BASE + 0x0000 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	MSTR	CONT_SCKE	DCONF		FRZ	MTFE	PCSSE	ROOE	0	0	PCSIS5	PCSIS4	PCSIS3	PCSIS2	PCSIS1	PCSIS0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	MDIS	DIS_TXF	DIS_RXF	CLR_TXF	CLR_RXF	SMPL_PT		0	0	0	0	0	0	0	HALT
W																
Reset	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Figure 30-3. DSPI Module Configuration Register (DSPI_MCR)

Table 30-3. DSPI_MCR Field Descriptions

Field	Description
MSTR	Master/Slave Mode Select. The MSTR bit configures the DSPI for either Master Mode or Slave Mode. 0 DSPI is in Slave Mode. 1 DSPI is in Master Mode.
CONT_SCKE	Continuous SCK Enable. The CONT_SCKE bit enables the Serial Communication Clock (SCK) to run continuously. See Section 30.4.9, Continuous Serial Communications Clock, for details. 0 Continuous SCK disabled. 1 Continuous SCK enabled.

Table 30-3. DSPI_MCR Field Descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>DCONF</td><td>DSPI Configuration. The DCONF field selects between the three different configurations of the DSPI. The values below list the DCONF values for the various configurations.
<table>
<tr><th>DCONF</th><th>DSPI Configuration</th></tr>
<tr><td>00</td><td>SPI</td></tr>
<tr><td>01</td><td>DSI</td></tr>
<tr><td>10</td><td>CSI</td></tr>
<tr><td>11</td><td>Reserved</td></tr>
</table></td></tr>
<tr><td>FRZ</td><td>Freeze. The FRZ bit enables the DSPI transfers to be stopped on the next frame boundary when the device enters Debug Mode.
0 Do not halt serial transfers.1Halt serial transfers.</td></tr>
<tr><td>MTFE</td><td>Modified Timing Format Enable. The MTFE bit enables a modified transfer format to be used. See Section 30.4.8.4, Modified SPI/DSI Transfer Format (MTFE = 1, CPHA = 1), for more information.
0 Modified SPI transfer format disabled.
1 Modified SPI transfer format enabled.</td></tr>
<tr><td>PCSSE</td><td>Peripheral Chip Select Strobe Enable. The PCSSE bit enables the PCS[5]/$\overline{\text{PCSS}}$ to operate as an PCS Strobe output signal. See Section 30.4.7.5, Peripheral Chip Select Strobe Enable (PCSS), for more information.
0 PCS[5]/$\overline{\text{PCSS}}$ is used as the Peripheral Chip Select[5] signal.
1 PCS[5]/$\overline{\text{PCSS}}$ is used as an active-low PCS Strobe signal.</td></tr>
<tr><td>ROOE</td><td>Receive FIFO Overflow Overwrite Enable. The ROOE bit enables an RX FIFO overflow condition to either ignore the incoming serial data or to overwrite existing data. If the RX FIFO is full and new data is received, the data from the transfer that generated the overflow is either ignored or shifted in to the shift register. If the ROOE bit is asserted, the incoming data is shifted in to the shift register. If the ROOE bit is negated, the incoming data is ignored. See Section 30.4.12.6, Receive FIFO Overflow Interrupt Request, for more information.
0 Incoming data is ignored.
1 Incoming data is shifted in to the shift register.</td></tr>
<tr><td>PCSISn</td><td>Peripheral Chip Select Inactive State. The PCSIS bit determines the inactive state of the PCS[x] signal.
0 The inactive state of PCS[x] is low.
1 The inactive state of PCS[x] is high.</td></tr>
<tr><td>MDIS</td><td>Module Disable. The MDIS bit allows the clock to be stopped to the non-memory mapped logic in the DSPI effectively putting the DSPI in a software controlled power-saving state. See Section 30.4.13, Power Saving Features, for more information. The reset value of the MDIS bit is parameterized, with a default reset value of ‘1’.
0 Enable DSPI clocks.
1 Allow external logic to disable DSPI clocks.</td></tr>
<tr><td>DIS_TXF</td><td>Disable Transmit FIFO. The DIS_TXF bit provides a mechanism to disable the TX FIFO. When the TX FIFO is disabled, the transmit part of the DSPI operates as a simplified double-buffered SPI. See Section 30.4.3.3, FIFO Disable Operation, for details.
0 TX FIFO is enabled.
1 TX FIFO is disabled.</td></tr>
</table>

Table 30-3. DSPI_MCR Field Descriptions (continued)

Field	Description
DIS_RXF	Disable Receive FIFO. The DIS_RXF bit provides a mechanism to disable the RX FIFO. When the RX FIFO is disabled, the receive part of the DSPI operates as a simplified double-buffered SPI. See Section 30.4.3.3, FIFO Disable Operation, for details. 0 RX FIFO is enabled. 1 RX FIFO is disabled.
CLR_TXF	Clear TX FIFO. CLR_TXF is used to flush the TX FIFO. Writing a '1' to CLR_TXF clears the TX FIFO Counter. The CLR_TXF bit is always read as zero. 0 Do not clear the TX FIFO Counter. 1 Clear the TX FIFO Counter.
CLR_RXF	Clear RX FIFO. CLR_RXF is used to flush the RX FIFO. Writing a '1' to CLR_RXF clears the RX Counter. The CLR_RXF bit is always read as zero. 0 Do not clear the RX FIFO Counter. 1 Clear the RX FIFO Counter.
SMPL_PT	SMPL_PT — Sample Point. SMPL_PT allows the host software to select when the DSPI Master samples SIN in Modified Transfer Format. Figure 30-31 shows where the master can sample the SIN pin. The table below lists the various delayed sample points.

SMPL_PT	Number of system clock cycles between odd-numbered edge of SCK*n* and sampling of SIN*n*.
00	0
01	1
10	2
11	Reserved

Field	Description
HALT	Halt. The HALT bit provides a mechanism by software to start and stop DSPI transfers. See Section 30.4.2, Start and Stop of DSPI Transfers, for details on the operation of this bit. 0 Start transfers. 1 Stop transfers.

30.3.2.2 DSPI Transfer Count Register (DSPI_TCR)

The DSPI_TCR contains a counter that indicates the number of SPI transfers made. The transfer counter is intended to assist in queue management.

NOTE

The user must not write to the DSPI_TCR while the DSPI is running.

Offset: DSPI_BASE + 0x0008 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	SPI_TCNT															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 30-4. DSPI Transfer Count Register (DSPI_TCR)

Table 30-4. DSPI_TCR Field Descriptions

Field	Description
SPI_TCNT	SPI Transfer Counter. SPI_TCNT is used to keep track of the number of SPI transfers made. The SPI_TCNT field counts the number of SPI transfers the DSPI makes. The SPI_TCNT field is incremented every time the last bit of a SPI frame is transmitted. A value written to SPI_TCNT presets the counter to that value. SPI_TCNT is reset to 0 at the beginning of the frame when the CTCNT field is set in the executing SPI command. The transfer counter wraps around; i.e., incrementing the counter past 65,535 (0xFFFF) resets the counter to 0.

30.3.2.3 DSPI Clock and Transfer Attributes Registers 0–7 (DSPI_CTAR*n*)

The DSPI_CTAR*n* registers are used to define different transfer attribute configurations. SPI and DSI transfers select one of the DSPI_CTAR*n* registers from which to get their transfer attributes. The user must not write to the DSPI_CTAR*n* registers while the DSPI is in the running state.

In master mode, the DSPI_CTAR*n* registers define combinations of transfer attributes such as frame size, clock phase and polarity, data bit ordering, baud rate, and various delays. In slave mode, a subset of the bitfields in the DSPI_CTAR0 and DSPI_CTAR1 registers are used to set the slave transfer attributes. See the individual bit descriptions for details on which bits are used in slave modes.

When the DSPI is configured as a SPI master, the CTAS field in the command portion of the TX FIFO entry selects which DSPI_CTAR*n* register is used. When the DSPI is configured as a SPI bus slave, the DSPI_CTAR0 register is used.

When the DSPI is configured as a DSI master, the DSICTAS field in the DSPI DSI Configuration Register (DSPI_DSICR) selects which DSPI_CTAR*n* register is used. For more information on the DSPI_DSICR, see Section 30.3.2.10, DSPI DSI Configuration Register (DSPI_DSICR). When the DSPI is configured as a DSI bus slave, the DSPI_CTAR1 register is used.

In CSI configuration, the transfer attributes are selected based on whether the current frame is SPI data or DSI data. SPI transfers in CSI configuration follow the protocol described for SPI configuration, and DSI transfers in CSI configuration follow the protocol described for DSI configuration. CSI configuration is only valid in conjunction with master mode. See Section 30.4.5, Combined Serial Interface (CSI) Configuration, for more details.

In continuous clock mode, only t_{DT} is supported for TSB. However, in TSB noncontinuous clock mode, both the PDT and DT delays are valid.

Offset: DSPI_BASE + 0x000C (DSPI_CTAR0), 0x0010 (DSPI_CTAR1), 0x0014 (DSPI_CTAR2), 0x0018 (DSPI_CTAR3), 0x001C (DSPI_CTAR4), 0x0020 (DSPI_CTAR5), 0x0024 (DSPI_CTAR6), 0x0028 (DSPI_CTAR7) — Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	DBR	FMSZ				CPOL	CPHA	LSB FE	PCSSCK		PASC		PDT		PBR	
Reset	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	CSSCK				ASC				DT				BR			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 30-5. DSPI Clock and Transfer Attributes Registers 0–7 (DSPI_CTAR*n*)

Table 30-5. DSPI_CTAR*n* Field Description

Field	Description
DBR	Double Baud Rate. The DBR bit doubles the effective baud rate of the Serial Communications Clock (SCK). This field is only used in master mode. It effectively halves the baud rate division ratio supporting faster frequencies and odd division ratios for the serial communications clock (SCK). When the DBR bit is set, the duty cycle of the SCK depends on the value in the baud rate prescaler and the clock phase bit as listed in Table 30-6. See the BR[0:3] field description for details on how to compute the baud rate. If the overall baud rate is divide by two or divide by three of the system clock then neither the continuous SCK enable or the modified timing format enable bits should be set. 0 The baud rate is computed normally with a 50/50 duty cycle. 1 The baud rate is doubled with the duty cycle depending on the Baud Rate Prescaler.
FMSZ	Frame Size. The FMSZ field selects the number of bits transferred per frame. The FMSZ field is used in Master Mode and Slave Mode. Table 30-7 lists the frame size encodings. When operating in TSB confirmation, detailed on Section 30.4.10, Timed Serial Bus (TSB), the FMSZ defines the point with in the 32-bit (maximum length) frame where control of the CS switches from the DSPI_DSICR to the DSPI_DSICR1 register. The cross over point must range between 4 bits and 16 bits and is encoded per Table 30-7. The remaining frame after the cross over point, regardless of how many bits are remaining, is controlled by the DSPI_DSICR1 register.
CPOL	Clock Polarity. The CPOL bit selects the inactive state of the Serial Communications Clock (SCK). This bit is used in both Master and Slave Mode. For successful communication between serial devices, the devices must have identical clock polarities. When the Continuous Selection Format is selected, switching between clock polarities without stopping the DSPI can cause errors in the transfer due to the peripheral device interpreting the switch of clock polarity as a valid clock edge. 0 The inactive state value of SCK is low. 1 The inactive state value of SCK is high.

Table 30-5. DSPI_CTAR*n* Field Description (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>CPHA</td><td>Clock Phase. The CPHA bit selects which edge of SCK causes data to change and which edge causes data to be captured. This bit is used in both master and slave mode. For successful communication between serial devices, the devices must have identical clock phase settings. Continuous SCK is only supported for CPHA = 1.
0 Data is captured on the leading edge of SCK and changed on the following edge.
1 Data is changed on the leading edge of SCK and captured on the following edge.</td></tr>
<tr><td>LSBFE</td><td>LSB First. The LSBFE bit selects if the LSB or MSB of the frame is transferred first. This bit is only used in master mode. When operating in TSB configuration, this bit should be always 1.
0 Data is transferred MSB first.
1 Data is transferred LSB first.</td></tr>
<tr><td>PCSSCK</td><td>PCS to SCK Delay Prescaler. The PCSSCK field selects the prescaler value for the delay between assertion of PCS and the first edge of the SCK. This field is only used in master mode. The table below lists the prescaler values. See the CSSCK[0:3] field description for details on how to compute the PCS to SCK delay.
<table>
<tr><th>PCSSCK</th><th>PCS to SCK Delay Prescaler Value</th></tr>
<tr><td>00</td><td>1</td></tr>
<tr><td>01</td><td>3</td></tr>
<tr><td>10</td><td>5</td></tr>
<tr><td>11</td><td>7</td></tr>
</table></td></tr>
<tr><td>PASC</td><td>After SCK Delay Prescaler. The PASC field selects the prescaler value for the delay between the last edge of SCK and the negation of PCS. This field is only used in master mode. The table below lists the prescaler values. See the ASC[0:3] field description for details on how to compute the After SCK Delay.
<table>
<tr><th>PASC</th><th>After SCK Delay Prescaler Value</th></tr>
<tr><td>00</td><td>1</td></tr>
<tr><td>01</td><td>3</td></tr>
<tr><td>10</td><td>5</td></tr>
<tr><td>11</td><td>7</td></tr>
</table></td></tr>
<tr><td>PDT</td><td>Delay after Transfer Prescaler. The PDT field selects the prescaler value for the delay between the negation of the PCS signal at the end of a frame and the assertion of PCS at the beginning of the next frame. The PDT field is only used in master mode. The table below lists the prescaler values. See the DT[0:3] field description for details on how to compute the delay after transfer.
<table>
<tr><th>PDT</th><th>Delay after Transfer Prescaler Value</th></tr>
<tr><td>00</td><td>1</td></tr>
<tr><td>01</td><td>3</td></tr>
<tr><td>10</td><td>5</td></tr>
<tr><td>11</td><td>7</td></tr>
</table></td></tr>
</table>

Table 30-5. DSPI_CTAR*n* Field Description (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>PBR</td><td>Baud Rate Prescaler. The PBR field selects the prescaler value for the baud rate. This field is only used in master mode. The baud rate is the frequency of the serial communications clock (SCK). The system clock is divided by the prescaler value before the baud rate selection takes place. The baud rate prescaler values are listed in the table below. See the BR[0:3] field description for details on how to compute the baud rate.
<table>
<tr><th>PBR</th><th>Baud Rate Prescaler Value</th></tr>
<tr><td>00</td><td>2</td></tr>
<tr><td>01</td><td>3</td></tr>
<tr><td>10</td><td>5</td></tr>
<tr><td>11</td><td>7</td></tr>
</table></td></tr>
<tr><td>CSSCK</td><td>PCS to SCK Delay Scaler. The CSSCK field selects the scaler value for the PCS to SCK delay. This field is only used in master mode. The PCS to SCK delay is the delay between the assertion of PCS and the first edge of the SCK. Table 30-8 list the scaler values.The PCS to SCK delay is a multiple of the system clock period and it is computed according to the following equation:
$$t_{CSC} = \frac{1}{f_{SYS}} \times PCSSCK \times CSSCK$$ ***Eqn. 30-1***
See Section 30.4.7.2, PCS to SCK Delay (tCSC), for more details.</td></tr>
<tr><td>ASC</td><td>After SCK Delay Scaler. The ASC field selects the scaler value for the after SCK delay. This field is only used in master mode. The after SCK delay is the delay between the last edge of SCK and the negation of PCS. Table 30-9 list the scaler values.The after SCK delay is a multiple of the system clock period, and it is computed according to the following equation:
$$t_{ASC} = \frac{1}{f_{SYS}} \times PASC \times ASC$$ ***Eqn. 30-2***
See Section 30.4.7.3, After SCK Delay (tASC), for more details.</td></tr>
<tr><td>DT</td><td>Delay after Transfer Scaler. The DT field selects the delay after transfer scaler. This field is only used in master mode. The delay after transfer is the time between the negation of the PCS signal at the end of a frame and the assertion of PCS at the beginning of the next frame. Table 30-10 lists the scaler values. In the continuous serial communications clock operation the DT value is fixed to one TSCK, except when the TSBC bit from DSPI_DSICR register is enabling the TSB configuration. See detailed information onSection 30.4.10, Timed Serial Bus (TSB). The delay after transfer is a multiple of the system clock period and it is computed according to the following equation:
$$t_{DT} = \frac{1}{f_{SYS}} \times PDT \times DT$$ ***Eqn. 30-3***
See Section 30.4.7.4, Delay after Transfer (tDT), for more details.</td></tr>
<tr><td>BR</td><td>Baud Rate Scaler. The BR field selects the scaler value for the baud rate. This field is only used in master mode. The pre-scaled system clock is divided by the baud rate scaler to generate the frequency of the SCK. Table 30-11 lists the baud rate scaler values.The baud rate is computed according to the following equation:
$$\text{SCK baud rate} = \frac{f_{SYS}}{PBR} \times \frac{1 + DBR}{BR}$$ ***Eqn. 30-4***
See Section 30.4.7.1, Baud Rate Generator, for more details.</td></tr>
</table>

Table 30-6. DSPI SCK Duty Cycle

DBR	CPHA	PBR	SCK Duty Cycle
0	any	any	50/50
1	0	00	50/50
1	0	01	33/66
1	0	10	40/60
1	0	11	43/57
1	1	00	50/50
1	1	01	66/33
1	1	10	60/40
1	1	11	57/43

Table 30-7. DSPI Transfer Frame Size

FMSZ	Framesize	FMSZ	Framesize
0000	Reserved	1000	9
0001	Reserved	1001	10
0010	Reserved	1010	11
0011	4	1011	12
0100	5	1100	13
0101	6	1101	14
0110	7	1110	15
0111	8	1111	16

Table 30-8. DSPI PCS to SCK Delay Scaler

CSSCK	PCS to SCK Delay Scaler Value	CSSCK	PCS to SCK Delay Scaler Value
0000	2	1000	512
0001	4	1001	1024
0010	8	1010	2048
0011	16	1011	4096
0100	32	1100	8192
0101	64	1101	16384
0110	128	1110	32768
0111	256	1111	65536

Table 30-9. DSPI After SCK Delay Scaler

ASC	After SCK Delay Scaler Value	ASC	After SCK Delay Scaler Value
0000	2	1000	512
0001	4	1001	1024
0010	8	1010	2048
0011	16	1011	4096
0100	32	1100	8192
0101	64	1101	16384
0110	128	1110	32768
0111	256	1111	65536

Table 30-10. DSPI Delay after Transfer Scaler

DT	Delay after Transfer Scaler Value	DT	Delay after Transfer Scaler Value
0000	2	1000	512
0001	4	1001	1024
0010	8	1010	2048
0011	16	1011	4096
0100	32	1100	8192
0101	64	1101	16384
0110	128	1110	32768
0111	256	1111	65536

Table 30-11. DSPI Baud Rate Scaler

BR	Baud Rate Scaler Value	BR	Baud Rate Scaler Value
0000	2	1000	256
0001	4	1001	512
0010	6	1010	1024
0011	8	1011	2048
0100	16	1100	4096
0101	32	1101	8192
0110	64	1110	16384
0111	128	1111	32768

30.3.2.4 DSPI Status Register (DSPI_SR)

The DSPI_SR contains status and flag bits. The bits reflect the status of the DSPI and indicate the occurrence of events that can generate interrupt or DMA requests. Software can clear a flag bit in the DSPI_SR by writing a 1 to it. Writing a 0 to a flag bit has no effect.

NOTE

This register cannot be written in module disable mode, owing to the use of power saving mechanisms.

NOTE

When generating DSPI bit frames in Continuous Peripheral Chip Select mode (DSPIx_PUSHR[CONT=1]) and when changing DSPIx_CTARn bit fields between frames, adhere to the following conditions, as they can generate error if:

- If DSPIx_CTARn[CPHA]=1, DSPIx_MCR[CONT_SCKE = 0], and DSPIx_CTARn[CPOL, CPHA, PCSSCK or PBR] change between frames.
- If DSPIx_CTARn[CPHA]=0 or DSPIx_MCR[CONT_SCKE = 1] and any bit field of DSPIx_CTARn changes between frames except DSPIx_CTARn[PBR].

Offset: DSPI_BASE + 0x002C — Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	TCF	TXRXS	0	EOQF	TFUF	0	TFFF	0	0	0	0	0	RFOF	0	RFDF	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	TXCTR				TXNXTPTR				RXCTR				POPNXTPTR			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 30-6. DSPI Status Register (DSPI_SR)

Table 30-12. DSPI_SR Field Descriptions

Field	Description
TCF	Transfer Complete Flag. The TCF bit indicates that all bits in a frame have been shifted out. The TCF bit is set at the end of the frame transfer. The TCF bit remains set until cleared by software. 0 Transfer not complete. 1 Transfer complete.
TXRXS	TX & RX Status. The TXRXS bit reflects the status of the DSPI. See Section 30.4.2, Start and Stop of DSPI Transfers, for information on how what causes this bit to be negated or asserted. 0 TX and RX operations are disabled (DSPI is in STOPPED state). 1 TX and RX operations are enabled (DSPI is in RUNNING state).
EOQF	End of Queue Flag. The EOQF bit indicates that transmission in progress is the last entry in a queue. The EOQF bit is set when TX FIFO entry has the EOQ bit set in the command halfword and the end of the transfer is reached. The EOQF bit remains set until cleared by software. When the EOQF bit is set, the TXRXS bit is automatically cleared. 0 EOQ is not set in the executing command. 1 EOQ bit is set in the executing SPI command.
TFUF	Transmit FIFO Underflow Flag. The TFUF bit indicates that an underflow condition in the TX FIFO has occurred. The transmit underflow condition is detected only for DSPI blocks operating in slave mode and SPI configuration. The TFUF bit is set when the TX FIFO of a DSPI operating in SPI slave mode is empty, and a transfer is initiated by an external SPI master. The TFUF bit remains set until cleared by software. 0 TX FIFO underflow has not occurred. 1 TX FIFO underflow has occurred.
TFFF	Transmit FIFO Fill Flag. The TFFF bit provides a method for the DSPI to request more entries to be added to the TX FIFO. The TFFF bit is set while the TX FIFO is not full. The TFFF bit can be cleared by host software or an acknowledgement from the DMA controller when the TX FIFO is full. 0 TX FIFO is full. 1 TX FIFO is not full.
RFOF	Receive FIFO Overflow Flag. The RFOF bit indicates that an overflow condition in the RX FIFO has occurred. The bit is set when the RX FIFO and shift register are full and a transfer is initiated. The bit remains set until cleared by software. 0 RX FIFO overflow has not occurred. 1 RX FIFO overflow has occurred.
RFDF	Receive FIFO Drain Flag. The RFDF bit provides a method for the DSPI to request that entries be removed from the RX FIFO. The bit is set while the RX FIFO is not empty. The RFDF bit can be cleared by host software or an acknowledgement from the DMA controller when the RX FIFO is empty. 0 RX FIFO is empty. 1 RX FIFO is not empty.
TXCTR	TX FIFO Counter. The TXCTR field indicates the number of valid entries in the TX FIFO. The TXCTR is incremented every time the DSPI _PUSHR is written. The TXCTR is decremented every time a SPI command is executed and the SPI data is transferred to the shift register.
TXNXTPTR	Transmit Next Pointer. The TXNXTPTR field indicates which TX FIFO Entry is transmitted during the next transfer. The TXNXTPTR field is updated every time SPI data is transferred from the TX FIFO to the shift register. See Section 30.4.3.4, Transmit First-In First-Out (TX FIFO) Buffering Mechanism, for more details.
RXCTR	RX FIFO Counter. The RXCTR field indicates the number of entries in the RX FIFO. The RXCTR is decremented every time the DSPI _POPR is read. The RXCTR is incremented every time data is transferred from the shift register to the RX FIFO.
POPNXTPTR	Pop Next Pointer. The POPNXTPTR field contains a pointer to the RX FIFO entry that is returned when the DSPI_POPR is read. The POPNXTPTR is updated when the DSPI_POPR is read. See Section 30.4.3.5, Receive First-In First-Out (RX FIFO) Buffering Mechanism, for more details.

30.3.2.5 DSPI DMA/Interrupt Request Select and Enable Register (DSPI_RSER)

The DSPI_RSER serves two purposes. It enables flag bits in the DSPI_SR to generate DMA requests or interrupt requests. The DSPI_RSER also selects the type of request to be generated. See the individual bit descriptions for information on the types of requests the bits support. The user must not write to the DSPI_RSER while the DSPI is in the running state.

Offset: DSPI_BASE + 0x0030 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	TCF_RE	0	0	EOQF_RE	TFUF_RE	0	TFFF_RE	TFFF_DIRS	0	0	0	0	RFOF_RE	0	RFDF_RE	RFDF_DIRS
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 30-7. DSPI DMA/Interrupt Request Select and Enable Register (DSPI_RSER)

Table 30-13. DSPI_RSER Field Descriptions

Field	Description
TCF_RE	Transmission Complete Request Enable. The TCF_RE bit enables TCF flag in the DSPI_SR to generate an interrupt request. 0 TCF interrupt requests are disabled. 1 TCF interrupt requests are enabled.
EOQF_ RE	DSPI Finished Request Enable. The EOQF_RE bit enables the EOQF flag in the DSPI_SR to generate an interrupt request. 0 EOQF interrupt requests are disabled. 1 EOQF interrupt requests are enabled.
TFUF_RE	Transmit FIFO Underflow Request Enable. The TFUF_RE bit enables the TFUF flag in the DSPI_SR to generate an interrupt request. 0 TFUF interrupt requests are disabled. 1 TFUF interrupt requests are enabled.
TFFF_RE	Transmit FIFO Fill Request Enable. The TFFF_RE bit enables the TFFF flag in the DSPI_SR to generate a request. The TFFF_DIRS bit selects between generating an interrupt request or a DMA requests. 0 TFFF interrupt requests or DMA requests are disabled. 1 TFFF interrupt requests or DMA requests are enabled.
TFFF_DIRS	Transmit FIFO Fill DMA or Interrupt Request Select. The TFFF_DIRS bit selects between generating a DMA request or an interrupt request. When the TFFF flag bit in the DSPI_SR is set, and the TFFF_RE bit in the DSPI_RSER register is set, this bit selects between generating an interrupt request or a DMA request. 0 Interrupt request is generated. 1 DMA request is generated.

Table 30-13. DSPI_RSER Field Descriptions (continued)

Field	Description
RFOF_RE	Receive FIFO Overflow Request Enable. The RFOF_RE bit enables the RFOF flag in the DSPI_SR to generate an interrupt requests. 0 RFOF interrupt requests are disabled. 1 RFOF interrupt requests are enabled.
RFDF_RE	Receive FIFO Drain Request Enable. The RFDF_RE bit enables the RFDF flag in the DSPI_SR to generate a request. The RFDF_DIRS bit selects between generating an interrupt request or a DMA request. 0 RFDF interrupt requests or DMA requests are disabled. 1 RFDF interrupt requests or DMA requests are enabled.
RFDF_DIRS	Receive FIFO Drain DMA or Interrupt Request Select. The RFDF_DIRS bit selects between generating a DMA request or an interrupt request. When the RFDF flag bit in the DSPI_SR is set, and the RFDF_RE bit in the DSPI_RSER register is set, the RFDF_DIRS bit selects between generating an interrupt request or a DMA request. 0 Interrupt request is generated. 1 DMA request is generated.

30.3.2.6 DSPI PUSH TX FIFO Register (DSPI_PUSHR)

The DSPI_PUSHR provides a means to write to the TX FIFO. Data written to this register is transferred to the TX FIFO. See Section 30.4.3.4, Transmit First-In First-Out (TX FIFO) Buffering Mechanism, for more information. Write accesses of 8 or 16 bits to the DSPI_PUSHR transfer 32 bits to the TX FIFO.

NOTE

Only the TXDATA field is used for DSPI slaves.

NOTE

When the DSPI module has more than one entry in the TX FIFO and only one entry is written and that entry has the CONT bit set, and continuous SCK clock selected the PCS levels may change between transfer complete and write of the next data to the DSPI_PUSHR register. To ensure PCS stability during data transmission in Continious Selection Format and Continious SCK clock enabled make sure that the data with reset CONT bit is written to DSPI_PUSHR register before previous data sub-frame (with CONT bit set) transfer is over.

Offset: DSPI_BASE + 0x0034 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CONT	CTAS			EOQ	CT CNT	0	0	0	0	PCS5	PCS4	PCS3	PCS2	PCS1	PCS0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	TXDATA															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 30-8. DSPI PUSH TX FIFO Register (DSPI_PUSHR)

Table 30-14. DSPI_PUSHR Field Descriptions

Field	Description
CONT	Continuous Peripheral Chip Select Enable. The CONT bit selects a Continuous Selection Format. The bit is used in SPI master mode. The bit enables the selected PCS signals to remain asserted between transfers. See Section 30.4.8.5, Continuous Selection Format, for more information. **Note:** To ensure PCS stability during data transmission in Continuous Selection Format (and Continuous SCK clock enabled) make sure that the data with reset CONT bit is written to DSPI_PUSHR register before previous data sub-frame (with CONT bit set) transfer is over. 0 Return Peripheral Chip Select signals to their inactive state between transfers. 1 Keep Peripheral Chip Select signals asserted between transfers.
CTAS	Clock and Transfer Attributes Select. The CTAS field selects which DSPI_CTAR*n* register is used to set the transfer attributes for the associated SPI frame. The field is only used in SPI master mode. In SPI slave mode, DSPI_CTAR0 is used. The table below shows how the CTAS values map to the DSPI_CTAR*n* registers.

CTAS	Use Clock and Transfer Attributes from	CTAS	Use Clock and Transfer Attributes from
000	DSPI_CTAR0	100	DSPI_CTAR4
001	DSPI_CTAR1	101	DSPI_CTAR5
010	DSPI_CTAR2	110	DSPI_CTAR6
011	DSPI_CTAR3	111	DSPI_CTAR7

EOQ	End Of Queue. The EOQ bit provides a means for host software to signal to the DSPI that the current SPI transfer is the last in a queue. At the end of the transfer the EOQF bit in the DSPI_SR is set. 0 The SPI data is not the last data to transfer. 1 The SPI data is the last data to transfer.
CTCNT	Clear SPI_TCNT. The CTCNT provides a means for host software to clear the SPI transfer counter. The CTCNT bit clears the SPI_TCNT field in the DSPI_TCR register. The SPI_TCNT field is cleared before transmission of the current SPI frame begins. 0 Do not clear SPI_TCNT field in the DSPI_TCR. 1 Clear SPI_TCNT field in the DSPI_TCR.

Table 30-14. DSPI_PUSHR Field Descriptions (continued)

Field	Description
PCS*n*	Peripheral Chip Select 0–7. The PCS bits select which PCS signals are asserted for the transfer. 0 Negate the PCS[*x*] signal. 1 Assert the PCS[*x*] signal.
TXDATA	Transmit Data. The TXDATA field holds SPI data to be transferred according to the associated SPI command.

30.3.2.7 DSPI POP RX FIFO Register (DSPI_POPR)

The DSPI_POPR provides a means to read the RX FIFO. See Section 30.4.3.5, Receive First-In First-Out (RX FIFO) Buffering Mechanism, for a description of the RX FIFO operations. 8- or 16-bit read accesses to the DSPI_POPR read from the RX FIFO and update the counter and pointer.

Offset: DSPI_BASE + 0x0038 Access: Read

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	RXDATA															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 30-9. DSPI POP RX FIFO Register (DSPI_POPR)

Table 30-15. DSPI_POPR Field Descriptions

Field	Description
RXDATA	Received Data. The RXDATA field contains the SPI data from the RX FIFO entry pointed to by the Pop Next Data Pointer.

30.3.2.8 DSPI Transmit FIFO Registers 0–15 (DSPI_TXFR*n*)

The DSPI_TXFR*n* registers provide visibility into the TX FIFO for debugging purposes. Each register is an entry in the TX FIFO. The registers are read-only and cannot be modified. Reading the DSPI_TXFR*n* registers does not alter the state of the TX FIFO.

Offset: DSPI_BASE+ Access: Read

0x003C (DSPI_TXFR0)
0x0040 (DSPI_TXFR1)
0x0044 (DSPI_TXFR2)
0x0048 (DSPI_TXFR3)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	TXCMD															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	TXDATA															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 30-10. DSPI Transmit FIFO Register 0–15 (DSPI_TXFR*n*)

Table 30-16. DSPI_TXFR*n* Field Descriptions

Field	Description
TXCMD	Transmit Command. The TXCMD field contains the command that sets the transfer attributes for the SPI data. See Section 30.3.2.6, DSPI PUSH TX FIFO Register (DSPI_PUSHR), for details on the command field.
TXDATA	Transmit Data. The TXDATA field contains the SPI data to be shifted out.

30.3.2.9 DSPI Receive FIFO Registers 0–3 (DSPI_RXFR*n*)

The DSPI_RXFR*n* registers provide visibility into the RX FIFO for debugging purposes. Each register is an entry in the RX FIFO. The DSPI_RXFR registers are read-only. Reading the DSPI_RXFR*n* registers does not alter the state of the RX FIFO.

Offset: DSPI_BASE+ Access: Read

0x007C (DSPI_RXFR0)
0x0080 (DSPI_RXFR1)
0x0084 (DSPI_RXFR2)
0x0088 (DSPI_RXFR3)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	RXDATA															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 30-11. DSPI Receive FIFO Registers 0–15 (DSPI_RXFR*n*)

Table 30-17. DSPI_RXFR*n* Field Description

Field	Description
RXDATA	Receive Data. The RXDATA field contains the received SPI data.

30.3.2.10 DSPI DSI Configuration Register (DSPI_DSICR)

The DSI Configuration Register selects various attributes associated with DSI and CSI configurations. The user must not write to the DSPI_DSICR while the DSPI is in the running state.

Offset: DSPI_BASE + 0x00BC Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	TSBC	TXSS	0	0	CID
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	DCONT	DSICTAS			0	0	0	0	0	0	DPCS5	DPCS4	DPCS3	DPCS2	DPCS1	DPCS0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 30-12. DSPI DSI Configuration Register (DSPI_DSICR)

Table 30-18. DSPI_DSICR Field Descriptions

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>TSBC</td><td>Timed Serial Bus Configuration. The TSBC bit enables the Timed Serial Bus configuration. This configuration allows 32-bit data to be used. It also allows T_{DT} to be programmable. See Section 30.4.10, Timed Serial Bus (TSB), for detailed information.
0 Timed Serial Bus configuration disabled.
1 Timed Serial Bus configuration enabled.
If this bit is disabled, the DSPI_DSICR1 register should not be used.</td></tr>
<tr><td>TXSS</td><td>Transmit Data Source Select. The TXSS bit selects the source of data to be serialized. The source can be either data from host Software written to the DSPI DSI Alternate Serialization Data Register (DSPI_ASDR), or Parallel Input pin states latched into the DSPI DSI Serialization Data Register (DSPI_SDR).
0 Source of serialized data is the DSPI_SDR.
1 Source of serialized data is the DSPI_ASDR.</td></tr>
<tr><td>CID</td><td>Change In Data Transfer Enable. The CID bit enables a change in serialization data to initiate a transfer. The bit is used in master mode in DSI and CSI configurations to control when to initiate transfers. When the CID bit is set, serialization is initiated when the current DSI data differs from the previous DSI data shifted out. The DSPI_COMPR register is compared with the DSPI_SDR or DSPI_ASDR register to detect a change in data. Refer to Section 30.4.4.5, DSI Transfer Initiation Control, for more information. When the TSBC bit is set, the CID bit is used for both DSICR and DSICR1 registers.</td></tr>
<tr><td>DCONT</td><td>DSI Continuous Peripheral Chip Select Enable. Enables the PCSx signals to remain asserted between transfers. The DCONT bit affects the PCS signals in DSI master mode only. See Section 30.4.8.5, Continuous Selection Format, for details.
0 Return peripheral chip select signals to their inactive state after transfer is complete.
1 Keep peripheral chip select signals asserted after transfer is complete.</td></tr>
<tr><td>DSICTAS</td><td>DSI Clock and Transfer Attributes Select. The DSICTAS field selects which DSPI_CTARn register is used to provide transfer attributes in DSI configuration. The DSICTAS field is used in DSI master mode. In DSI slave mode, the DSPI_CTAR1 is always selected. The table below shows how the DSICTAS values map to the DSPI_CTARn registers. When TSB configuration is selected the DSICTAS bits control all 32 bits.
<table>
<tr><th>DSICTAS</th><th>DSI Clock and Transfer Attributes Controlled by</th><th>DSICTAS</th><th>DSI Clock and Transfer Attributes Controlled by</th></tr>
<tr><td>000</td><td>DSPI_CTAR0</td><td>100</td><td>DSPI_CTAR4</td></tr>
<tr><td>001</td><td>DSPI_CTAR1</td><td>101</td><td>DSPI_CTAR5</td></tr>
<tr><td>010</td><td>DSPI_CTAR2</td><td>110</td><td>DSPI_CTAR6</td></tr>
<tr><td>011</td><td>DSPI_CTAR3</td><td>111</td><td>DSPI_CTAR7</td></tr>
</table></td></tr>
<tr><td>DPCSn</td><td>DSI Peripheral Chip Select n. The DPCS bits select which PCS signals to assert during a DSI transfer. The DPCS bits only control the assertions of the PCS signals in DSI master mode.When TSB configuration is enabled, the DPCS bits only apply for the first 16 bits of the frame; the PCS used for any further bits is selected in the DSICR1 register.
0 Negate PCS[x].
1 Assert PCS[x].</td></tr>
</table>

30.3.2.11 DSPI DSI Serialization Data Register (DSPI_SDR)

The DSPI_SDR contains the signal states of the parallel input signals. The pin states of the parallel input signals are latched into the DSPI_SDR on the rising edge of every system clock. The DSPI_SDR is read-only. When the TXSS bit in the DSPI_DSICR is negated, the data in the DSPI_SDR is the source of the serialized data.

The DSPI_SDR is a 32-bit register. The upper 16 bits are only used when TSB is enabled. For non-TSB configurations, only the least 16 significant bits are used.

Offset: DSPI_BASE + 00C0 Access: Read

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	SER_DATA															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	SER_DATA															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 30-13. DSPI DSI Serialization Data Register (DSPI_SDR)

Table 30-19. DSPI_SDR Field Description

Bits	Description
SER_DATA	Serialized Data. The SER_DATA field contains the signal states of the Parallel Input signals.

30.3.2.12 DSPI DSI Alternate Serialization Data Register (DSPI_ASDR)

The DSPI_ASDR provides a means for host software to write the data to be serialized. When the TXSS bit in the DSPI_DSICR is set, the data in the DSPI_ASDR is the source of the serialized data. Writes to the DSPI_ASDR take effect on the next frame boundary.

The DSPI_ASDR is a 32-bit register. The upper 16 bits are only used when TSB is enabled. For non-TSB configurations, only the least 16 significant bits are used.

Offset: DSPI_BASE + 0x00C4 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	ASER_DATA															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	ASER_DATA															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 30-14. DSPI DSI Alternate Serialization Data Register (DSPI_ASDR)

Table 30-20. DSPI_ASDR Field Description

Field	Description
ASER_DATA	Alternate Serialized Data. The ASER_DATA field holds the alternate data to be serialized.

30.3.2.13 DSPI DSI Transmit Comparison Register (DSPI_COMPR)

The DSPI_COMPR holds a copy of the last transmitted DSI data. The DSPI_COMPR is read-only. DSI data is transferred to this register as it is loaded into the TX Shift Register.

The DSPI_COMPR is a 32-bit register. The upper 16 bits are only used when TSB is enabled. For non-TSB configurations only the least 16 significant bits are used.

Offset: DSPI_BASE + 0x00C8 Access: Read

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	COMP_DATA															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	COMP_DATA															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 30-15. DSPI DSI Transmit Comparison Register (DSPI_COMPR)

Table 30-21. DSPI_COMPR Field Description

Field	Description
COMP_DATA	Compare Data. The COMP_DATA field holds the last serialized DSI data.

30.3.2.14 DSPI DSI Deserialization Data Register (DSPI_DDR)

The DSPI_DDR register holds the signal states for the parallel output signals. The DSPI_DDR is read-only and it is memory mapped so that host software can read the incoming DSI frames.

Offset: DSPI_BASE + 0x00CC Access: Read

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	DESER_DATA															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	DESER_DATA															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 30-16. DSPI Deserialization Data Register (DSPI_DDR)

Table 30-22. DSPI_DDR Field Description

Field	Description
DESER_DATA	Deserialized Data. When TSB configuration is set, the DESER_DATA field holds deserialized data that is presented as signal states to the parallel output signals. If TSB is disabled, these bits are ignored, and only the lower 16 bits are valid.
DESER_DATA	Deserialized Data. The DESER_DATA field holds deserialized data that is presented as signal states to the parallel output signals.

30.3.2.15 DSPI DSI Configuration Register 1 (DSPI_DSICR1)

The DSI Configuration Register 1 selects various attributes associated with TSB configuration. The user must not write to the DSPI_DSICR1 while the DSPI is in the running state. If TSB configuration is not used, the register value is ignored.

Address: DSPI_BASE + 0x00D0 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	TSBCNT					0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	DPCS1_5	DPCS1_4	DPCS1_3	DPCS1_2	DPCS1_1	DPCS1_0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 30-17. DSPI DSI Configuration Register 1 (DSPI_DSICR1)

Table 30-23. DSPI_SDR Field Descriptions

Field	Description
TSBCNT	Timed Serial Bus Operation Count. When TSBC is set, TSBCNT defines the length of the TSB frame. A number between 4 and 32. The TSBCNT field selects number of bits to be shifted out during a transfer in TSB operation. The field sets the number of SCK cycles that the bus master generates to complete the transfer. The number of SCK cycles used is one more than the value in the TSBCNT field. The number of SCK cycles defined by TSBCNT must be equal to or greater than the frame size.
DPCS1_*x*	DSI Peripheral Chip Select 0–7. These bits define the CS to assert for the second part of the DSI frame when operating in TSB configuration with dual receiver. The DPCS1 bits select which of the PCS signals to assert during the second DSI transfer. The DPCS1 bits only control the assertions of the PCS signals in DSI master mode when in TSB configuration. 0 Negate PCS[*x*]. 1 Assert PCS[*x*].

30.4 Functional Description

The DSPI supports full-duplex, synchronous serial communications between the MCU and peripheral devices. The DSPI can also be used to reduce the number of pins required for I/O by serializing and deserializing as many as 16 parallel input/output signals from the eMIOS. All communications are through an SPI-like protocol. Specifically in the TSB configuration, the DSPI can serialize as many as 32 parallel input signals or 32 registered bits.

The DSPI has three configurations:

- SPI configuration, in which the DSPI operates as a basic SPI or a queued SPI.
- DSI configuration, in which the DSPI serializes and deserializes parallel input/output signals or bits from memory-mapped registers.
- CSI configuration, in which the DSPI combines the functionality of the SPI and DSI configurations.

The DCONF field in the DSPI_MCR register determines the DSPI configuration. See Table 30-3 for the DSPI configuration values.

The DSPI_CTAR*n* registers hold clock and transfer attributes. The manner in which a CTAR is selected depends on the DSPI configuration (SPI, DSI, or CSI). The SPI configuration can select which CTAR to use on a frame-by-frame basis by setting the CTAS field in the DSPI_PUSHR. The DSI configuration statically selects which CTAR to use. In CSI configuration, priority logic determines if SPI data or DSI data is transferred. The type of data transferred (whether DSI or SPI) dictates which CTAR the CSI configuration uses. See Section 30.3.2.3, DSPI Clock and Transfer Attributes Registers 0–7 (DSPI_CTARn), for information on DSPI_CTAR*n* fields.

The 16-bit shift register in the master and the 16-bit shift register in the slave are linked by the SOUT and SIN signals to form a distributed 32-bit register. The master and slave use 16-bit shift registers regardless the TSBC bit is asserted in the DSPI_DSICR register. When a data transfer operation is performed, data is serially shifted a pre-determined number of bit positions. Because the registers are linked, data is exchanged between the master and the slave; the data that was in the master’s shift register is now in the

shift register of the Slave, and vice versa. At the end of a transfer, the TCF bit in the DSPI_SR is set to indicate a completed transfer. Figure 30-18 illustrates how master and slave data is exchanged.

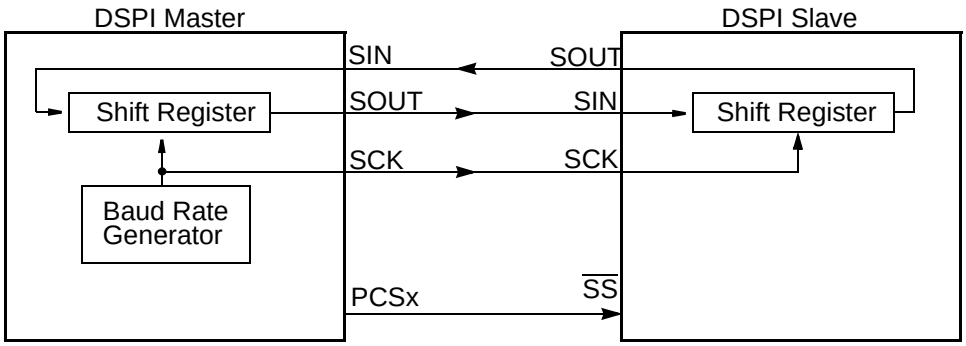


Figure 30-18. SPI and DSI Serial Protocol Overview

The DSPI has six peripheral chip select (PCS) signals that are used to select the slaves with which the DSPI communicates.

The three DSPI configurations share transfer protocol and timing properties so they are described independently of the configuration in Section 30.4.8, Transfer Formats. The transfer rate and delay settings are described in Section 30.4.7, DSPI Baud Rate and Clock Delay Generation.

See Section 30.4.13, Power Saving Features, for information on the power-saving features of the DSPI.

30.4.1 Modes of Operation

The DSPI modules have the following modes available:

- Master mode
- Slave mode
- Module disable mode
- Halt mode
- Debug mode

Master, slave, and module disable modes are module-specific modes. External halt and debug mode are device-specific modes.

The module-specific modes are determined by bits in the DSPI_MCR. External halt and debug mode are modes that the entire MCU can enter in parallel with the DSPI being configured in one of its block-specific modes.

30.4.1.1 Master Mode

In master mode, the DSPI can initiate communications with peripheral devices. The DSPI operates as bus master when the MSTR bit in the DSPI_MCR is set. The serial communications clock (SCK) is controlled by the master DSPI. All three DSPI configurations are valid in master mode.

In SPI configuration, master mode transfer attributes are controlled by the SPI command in the current TX FIFO entry. The CTAS field in the SPI command selects which of the eight DSPI_CTAR*n* registers are used to set the transfer attributes. Transfer attribute control is on a frame by frame basis. See Section 30.4.3, Serial Peripheral Interface (SPI) Configuration, for more details.

In DSI configuration, master mode transfer attributes are controlled by the DSPI DSI Configuration Register (DSPI_DSICR). A detailed description of the DSPI_DSCIR is located in Section 30.3.2.10, DSPI DSI Configuration Register (DSPI_DSICR). The DSISCTAS field in the DSPI_DSICR selects which of the DSPI_CTAR*n* registers are used to set the transfer attributes. Transfer attributes are set up during initialization and should not be changed between frames. See Section 30.4.4, Deserial Serial Interface (DSI) Configuration, for more details.

The CSI configuration is only available in master mode. In CSI configuration, the DSI data is transferred using DSI configuration transfer attributes and SPI data is transferred using the SPI configuration transfer attributes. In order for the bus slave to distinguish between DSI and SPI frames, the transfer attributes for the two types of frames must utilize different peripheral chip select signals. See Section 30.4.5, Combined Serial Interface (CSI) Configuration, for details.

30.4.1.2 Slave Mode

In slave mode, the DSPI responds to transfers initiated by a SPI master. The DSPI operates as bus slave when the MSTR bit in the DSPI_MCR register is negated. The DSPI slave is selected by a bus master by having the slave's $\overline{SS}$ asserted. In slave mode, the SCK is provided by the bus master. All transfer attributes are controlled by the bus master. However, clock polarity, clock phase, and numbers of bits to transfer must still be configured in the DSPI slave for proper communications.

The SPI and DSI configurations are valid in slave mode. In SPI slave mode, the slave transfer attributes are set in the DSPI_CTAR0. In DSI slave mode, the slave transfer attributes are set in the DSPI_CTAR1. In both SPI and DSI configurations, the DSPI in slave mode transfers data MSB first. The LSBFE field of the associated CTAR is ignored.

30.4.1.3 Module Disable Mode

The module disable mode is used for MCU power management. The clock to the non-memory mapped logic in the DSPI can be stopped while in module disable mode.The DSPI enters the module disable mode when the MDIS bit in DSPI_MCR is set. Logic external to the DSPI is needed to implement the module disable mode. See Section 30.4.13, Power Saving Features, for more details on the module disable mode.

30.4.1.4 Halt Mode

When the appropriate bit in the SIU_HLT0 register is set, a request to enter halt mode is sent to the DSPI. The DSPI does not acknowledge the request to enter halt mode until it has reached a frame boundary. When the DSPI has reached a frame boundary, it halts all operations and indicates that it is ready to have its clocks shut off. The DSPI exits halt mode and resumes normal operation once the clocks are turned on. Serial communications or register accesses made while in halt mode are ignored even if the clocks have not been shut off yet. See Section 30.4.13, Power Saving Features, for more details on the halt mode.

30.4.1.5 Debug Mode

The debug mode is used for system development and debugging. If the MCU enters debug mode while the FRZ bit in the DSPI_MCR is set, the DSPI stops all serial transfers and enters a stopped state. If the MCU enters debug mode while the FRZ bit is negated, the DSPI behavior is unaffected and remains dictated by the module-specific mode and configuration of the DSPI. The DSPI enters debug mode when a debug request is asserted by an external controller. See Figure 30-19 for a state diagram.

30.4.2 Start and Stop of DSPI Transfers

The DSPI has two operating states: stopped and running. The states are independent of DSPI configuration. The default state of the DSPI is stopped. In the stopped state, no serial transfers are initiated in master mode and no transfers are responded to in slave mode. The stopped state is also a safe state for writing the various configuration registers of the DSPI without causing undetermined results. The TXRXS bit in the DSPI_SR is negated in this state. In the running state, serial transfers take place. The TXRXS bit in the DSPI_SR is asserted in the running state. Figure 30-19 shows a state diagram of the start and stop mechanism. The transitions are described in Table 30-24.

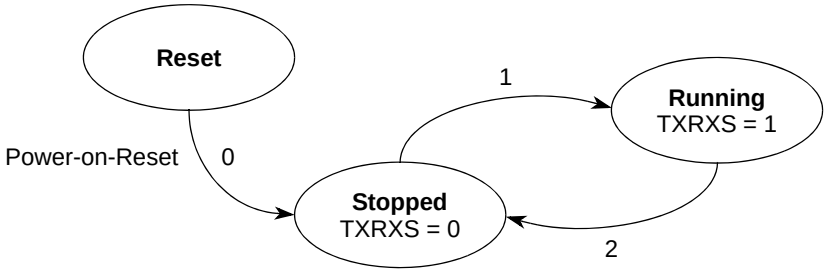


Figure 30-19. DSPI Start and Stop State Diagram

Table 30-24. State Transitions for Start and Stop of DSPI Transfers

Transition #	Current State	Next State	Description
0	Reset	Stopped	Generic power-on-reset transition
1	Stopped	Running	The DSPI is started (DSPI transitions to running) when all of the following conditions are true: • EOQF bit is clear • Debug mode is unselected or the FRZ bit is clear • HALT bit is clear
2	Running	Stopped	The DSPI stops (transitions from running to stopped) after the current frame for any one of the following conditions: • EOQF bit is set • Debug mode is selected and the FRZ bit is set • HALT bit is set

State transitions from running to stopped occur on the next frame boundary if a transfer is in progress, or on the next system clock cycle if no transfers are in progress.

30.4.3 Serial Peripheral Interface (SPI) Configuration

The SPI configuration transfers data serially using a shift register and a selection of programmable transfer attributes. The DSPI is in SPI configuration when the DCONF field in the DSPI_MCR is 0b00. The SPI frames can be from 4 to 16 bits long. The data to be transmitted can come from queues stored in RAM external to the DSPI. Host software or a DMA controller can transfer the SPI data from the queues to a first-in first-out (FIFO) buffer. The received data is stored in entries in the receive FIFO (RX FIFO) buffer. Host software or a DMA controller can transfer the received data from the RX FIFO to memory external to the DSPI. The FIFO buffer operations are described in Section 30.4.3.4, Transmit First-In First-Out (TX FIFO) Buffering Mechanism, and Section 30.4.3.5, Receive First-In First-Out (RX FIFO) Buffering Mechanism. The interrupt and DMA request conditions are described in Section 30.4.12, DMA and Interrupt Conditions.

Figure 30-20 shows an example of how a master DSPI connects to a SPI slave in SPI configuration.

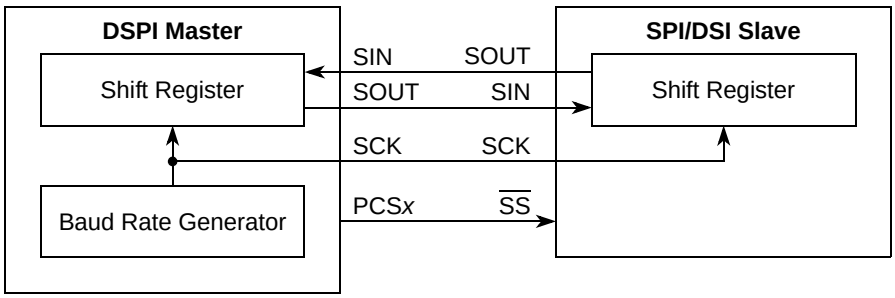


Figure 30-20. DSPI Connections for SPI and DSI Transfers

The SPI configuration supports two block-specific modes: master mode and slave mode. The FIFO operations are similar for the master mode and slave mode. The main difference is that in master mode the DSPI initiates and controls the transfer according to the fields in the SPI command field of the TX FIFO entry. In slave mode, the DSPI only responds to transfers initiated by a bus master external to the DSPI and the SPI command field of the TX FIFO entry is ignored.

30.4.3.1 SPI Master Mode

In SPI master mode, the DSPI initiates the serial transfers by controlling the serial communications clock (SCK) and the peripheral chip select (PCS) signals. The SPI command field in the executing TX FIFO entry determines which CTAR registers are used to set the transfer attributes and which PCS signal to assert. The command field also contains various bits that help with queue management and transfer protocol. See Section 30.3.2.6, DSPI PUSH TX FIFO Register (DSPI_PUSHR), for details on the SPI command fields. The data field in the executing TX FIFO entry is loaded into the shift register and shifted out on the serial out (SOUT) pin. In SPI master mode, each SPI frame to be transmitted has a command associated with it allowing for transfer attribute control on a frame by frame basis.

30.4.3.2 SPI Slave Mode

In SPI slave mode, the DSPI responds to transfers initiated by a SPI bus master. The DSPI does not initiate transfers. Certain transfer attributes such as clock polarity, clock phase and frame size must be set for

successful communication with a SPI master. The SPI slave mode transfer attributes are set in the DSPI_CTAR0.

30.4.3.3 FIFO Disable Operation

The FIFO disable mechanisms allow SPI transfers without using the TX FIFO or RX FIFO. The DSPI operates as a double-buffered simplified SPI when the FIFOs are disabled. The TX and RX FIFOs are disabled separately. The TX FIFO is disabled by writing a ‘1’ to the DIS_TXF bit in the DSPI_MCR. The RX FIFO is disabled by writing a ‘1’ to the DIS_RXF bit in the DSPI_MCR.

The FIFO disable mechanisms are transparent to the user and to host software; Transmit data and commands are written to the DSPI_PUSHR and received data is read from the DSPI_POPR. When the TX FIFO is disabled the TFFF, TFUF and TXCTR fields in DSPI_SR behave as if there is a one-entry FIFO but the contents of the DSPI_TXFR registers and TXNXTPTR are undefined. When the RX FIFO is disabled the RFDF, RFOF and RXCTR fields in the DSPI_SR behave as if there is a one-entry FIFO but the contents of the DSPI_RXFR registers and POPNXTPTR are undefined.

The TX and RX FIFOs must be disabled only if the application’s operating mode requires the FIFO to be disabled. A FIFO must be disabled before it is accessed. Failure to disable a FIFO prior to a first FIFO access is not supported, and may result in incorrect results.

30.4.3.4 Transmit First-In First-Out (TX FIFO) Buffering Mechanism

The TX FIFO functions as a buffer of SPI data and SPI commands for transmission. The TX FIFO holds four entries, each consisting of a command field and a data field. SPI commands and data are added to the TX FIFO by writing to the DSPI PUSH TX FIFO Register (DSPI_PUSHR). For more information on DSPI_PUSHR, refer to Section 30.3.2.6, DSPI PUSH TX FIFO Register (DSPI_PUSHR). TX FIFO entries can only be removed from the TX FIFO by being shifted out or by flushing the TX FIFO.

The TX FIFO counter field (TXCTR) in the DSPI Status Register (DSPI_SR) indicates the number of valid entries in the TX FIFO. The TXCTR is updated every time the DSPI _PUSHR is written or SPI data is transferred into the shift register from the TX FIFO. For more information on DSPI_SR, refer to Section 30.3.2.4, DSPI Status Register (DSPI_SR).

The TXNXTPTR field indicates which TX FIFO entry is transmitted during the next transfer. The TXNXTPTR contains the positive offset from DSPI_TXFR0 in number of 32-bit registers. For example, TXNXTPTR = 0b0010 (2) means that the DSPI_TXFR2 contains the SPI data and command for the next transfer. The TXNXTPTR field is incremented every time SPI data is transferred from the TX FIFO to the shift register.

30.4.3.4.1 Filling the TX FIFO

Host software or the eDMA controller can add (push) entries to the TX FIFO by writing to the DSPI_PUSHR. When the TX FIFO is not full, the TX FIFO fill flag (TFFF) in the DSPI_SR is set. The TFFF bit is cleared when TX FIFO is full and the eDMA controller indicates that a write to DSPI_PUSHR is complete or by host software writing a ‘1’ to the TFFF in the DSPI_SR. The TFFF can generate a DMA request or an interrupt request. See Section 30.4.12.2, Transmit FIFO Fill Interrupt or DMA Request (TFFF), for details.

The DSPI ignores attempts to push data to a full TX FIFO, i.e., the state of the TX FIFO is unchanged. No error condition is indicated.

30.4.3.4.2 Draining the TX FIFO

The TX FIFO entries are removed (drained) by shifting SPI data out through the shift register. Entries are transferred from the TX FIFO to the shift register and shifted out as long as there are valid entries in the TX FIFO. Every time an entry is transferred from the TX FIFO to the shift register, the TX FIFO counter is decremented by one. At the end of a transfer, the TCF bit in the DSPI_SR is set to indicate the completion of a transfer. The TX FIFO is flushed by writing a ‘1’ to the CLR_TXF bit in DSPI_MCR.

If an external bus master initiates a transfer with a DSPI slave while the slave’s DSPI TX FIFO is empty, the Transmit FIFO Underflow Flag (TFUF) in the slave’s DSPI_SR is set. See Section 30.4.12.4, Transmit FIFO Underflow Interrupt Request (TFUF), for details.

30.4.3.5 Receive First-In First-Out (RX FIFO) Buffering Mechanism

The RX FIFO functions as a buffer for data received on the SIN pin. The RX FIFO holds four received SPI data frames. SPI data is added to the RX FIFO at the completion of a transfer when the received data in the shift register is transferred into the RX FIFO. SPI data are removed (popped) from the RX FIFO by reading the DSPI POP RX FIFO Register (DSPI_POPR). RX FIFO entries can only be removed from the RX FIFO by reading the DSPI_POPR or by flushing the RX FIFO. For more information on the DSPI_POPR, refer to Section 30.3.2.7, DSPI POP RX FIFO Register (DSPI_POPR).

The RX FIFO Counter field (RXCTR) in the DSPI Status Register (DSPI_SR) indicates the number of valid entries in the RX FIFO. The RXCTR is updated every time the DSPI _POPR is read or SPI data is copied from the shift register to the RX FIFO.

The POPNXTPTR field in the DSPI_SR points to the RX FIFO entry that is returned when the DSPI_POPR is read. The POPNXTPTR contains the positive offset from DSPI_RXFR0 in number of 32-bit registers. For example, POPNXTPTR = 0b0010 means that the DSPI_RXFR2 contains the received SPI data that is returned when DSPI_POPR is read. The POPNXTPTR field is incremented every time the DSPI_POPR is read.

30.4.3.5.1 Filling the RX FIFO

The RX FIFO is filled with the received SPI data from the shift register. While the RX FIFO is not full, SPI frames from the shift register are transferred to the RX FIFO. Every time a SPI frame is transferred to the RX FIFO, the RX FIFO counter is incremented by one.

If the RX FIFO and shift register are full and a transfer is initiated, the RFOF bit in the DSPI_SR is asserted indicating an overflow condition. Depending on the state of the ROOE bit in the DSPI_MCR, the data from the transfer that generated the overflow is either ignored or shifted in to the shift register. If the ROOE bit is asserted, the incoming data is shifted in to the shift register. If the ROOE bit is negated, the incoming data is ignored.

30.4.3.5.2 Draining the RX FIFO

Host software or the eDMA controller can remove (pop) entries from the RX FIFO by reading the DSPI POP RX FIFO Register (DSPI_POPR). For more information on DSPI_POPR, refer to Section 30.3.2.7, DSPI POP RX FIFO Register (DSPI_POPR). A read of the DSPI_POPR decrements the RX FIFO counter by one. Attempts to pop data from an empty RX FIFO are ignored, and the RX FIFO counter remains unchanged. The data returned from reading an empty RX FIFO is undetermined.

When the RX FIFO is not empty, the RX FIFO drain flag (RFDF) in the DSPI_SR is set. The RFDF bit is cleared when the RX_FIFO is empty and the DMA controller indicates that a read from DSPI_POPR is complete or by host software writing a 1 to the RFDF.

30.4.4 Deserial Serial Interface (DSI) Configuration

The DSI configuration supports pin-count reduction by serializing parallel input signals or register bits and shifting them out in a SPI-like protocol. The timing and transfer protocol is described in Section 30.4.8, Transfer Formats. The received serial frames are converted to a parallel form (deserialized) and placed on the parallel output signals or in a register.

The various features of the DSI configuration are set in the DSPI_DSICR. For more information on the DSPI_DSICR, refer to Section 30.3.2.10, DSPI DSI Configuration Register (DSPI_DSICR). The DSPI is in DSI configuration when the DCONF field in the DSPI_MCR = 0b01.

The DSI frames can be from 4 to 16 bits long, but 4 to 32 bits can be used in the TSB configuration (see Section 30.4.10, Timed Serial Bus (TSB), for detailed information).

30.4.4.1 DSI Master Mode

In DSI master mode, the DSPI initiates and controls the DSI transfers. The DSI master has these conditions that can initiate a transfer:

- Continuous
- Change in data

The two transfer initiation conditions are described in Section 30.4.4.5, DSI Transfer Initiation Control. Transfer attributes are set during initialization. The DSICTAS field in the DSPI_DSICR determines which DSPI_CTAR*n* register controls the transfer attributes.

30.4.4.2 DSI Slave Mode

In DSI slave mode, the DSPI responds to transfers initiated by a SPI or DSI bus master. In this mode the DSPI does not initiate DSI transfers. Certain transfer attributes such as clock polarity and phase must be set for successful communication with a DSI master. The DSI slave mode transfer attributes are set in the DSPI_CTAR1.

30.4.4.3 DSI Serialization

In the DSI configuration, 4 to 16 bits can be serialized using two different sources. The TXSS bit in the DSPI_DSICR selects between the DSPI DSI Serialization Data Register (DSPI_SDR) and the DSPI DSI

Alternate Serialization Data Register (DSPI_ASDR) as the source of the serialized data. See Section 30.3.2.11, DSPI DSI Serialization Data Register (DSPI_SDR), and Section 30.3.2.12, DSPI DSI Alternate Serialization Data Register (DSPI_ASDR), for more details. The DSPI_SDR holds the latest parallel input signal values, which are sampled at every rising edge of the system clock. The DSPI_ASDR register is written by host software and used as an alternate source of serialized data.

A copy of the last 32-bit DSI frame shifted out of the Shift Register is stored in the DSPI DSI Transmit Comparison Register (DSPI_COMPR). This register provides added visibility for debugging and it serves as a reference for transfer initiation control.Section 30.3.2.13, DSPI DSI Transmit Comparison Register (DSPI_COMPR), contains details on the DSPI_COMPR. Figure 30-21 shows the DSI Serialization logic.

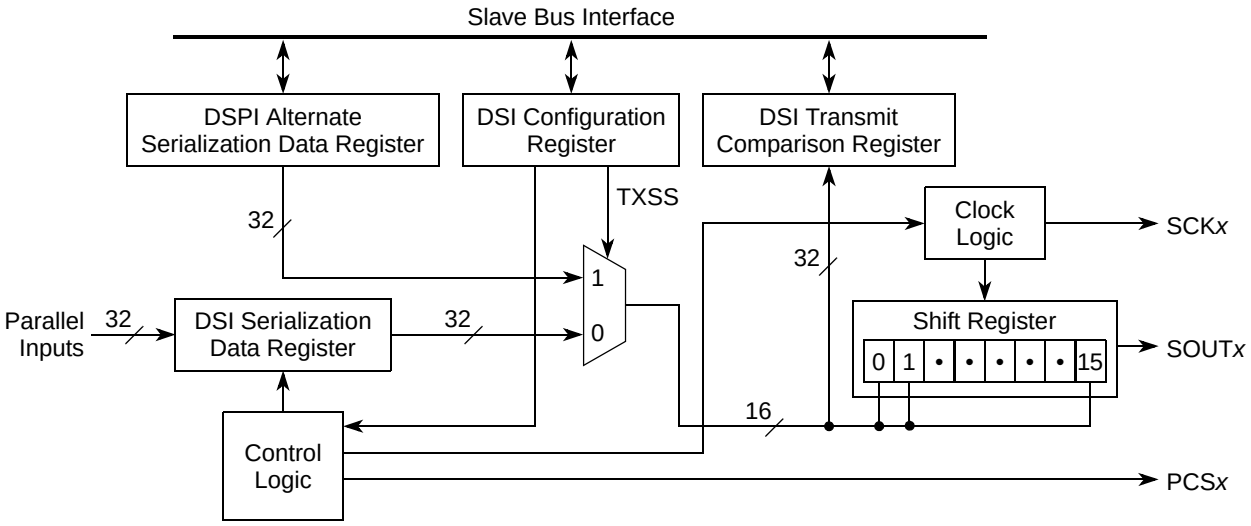
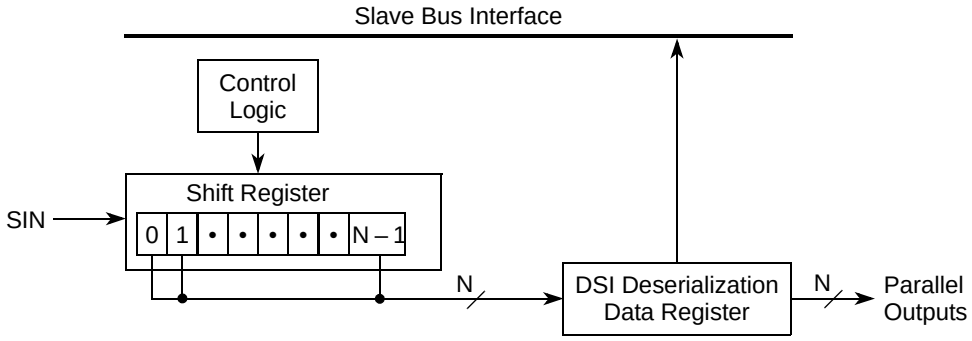


Figure 30-21. DSI Serialization Diagram

30.4.4.4 DSI Deserialization

When all bits in a DSI frame have been shifted in, the frame is copied to the DSPI DSI Deserialization Data Register (DSPI_DDR). This register presents the deserialized data as parallel output signal values. The DSPI_DDR is memory mapped to allow host software to read the deserialized data directly. For more information on the DSPI_DDR, refer to Section 30.3.2.14, DSPI DSI Deserialization Data Register (DSPI_DDR). Figure 30-22 shows the DSI Deserialization logic.

When all bits in a DSI frame have been shifted in, the frame is copied to the DSPI_DDR. This register presents the deserialized data as parallel output signal values. The DSPI_DDR is memory mapped to allow host software to read the deserialized data directly. For more information on the DSPI_DDR, refer to Section 30.3.2.14, DSPI DSI Deserialization Data Register (DSPI_DDR). Figure 30-22 shows the DSI deserialization logic.



In TSB configuration, the number of bits N = 32. For non-TSB, N = 16.

Figure 30-22. DSI Deserialization Diagram

30.4.4.5 DSI Transfer Initiation Control

Data transfers for a master DSPI in DSI configuration are initiated by a condition. The transfer initiation conditions are selected by the TRRE and CID bits in the DSPI_DSICR. Table 30-25 lists the transfer initiation conditions.

Table 30-25. DSI Data Transfer Initiation Control

DSPI_DSICR Bits		Transfer Initiation Control
TRRE	CID	
0	0	Continuous
0	1	Change in Data

30.4.4.5.1 Continuous Control

For continuous control, the initiation of a transfer is based on the baud rate at which data is transferred between the DSPI and the external device. The baud rate is set in the DSPI_CTAR*n* register selected by the DSICTAS field in the DSPI_DSICR. A new DSI frame shifts out when the previous transfer cycle has completed and the delay after transfer (t_{DT}) has elapsed.

30.4.4.5.2 Change In Data Control

For change in data control, a transfer is initiated when the data to be serialized has changed since the transfer of the last DSI frame. A copy of the previously transferred DSI data is stored in the DSPI_COMPR. When the data in the DSPI_SDR or the DSPI_ASDR is different from the data in the DSPI_COMPR, a new DSI frame is transmitted. The TXSS bit in the DSPI_DSICR selects the register to which the DSPI_COMPR is compared.

30.4.5 Combined Serial Interface (CSI) Configuration

The CSI configuration of the DSPI is used to support SPI and DSI functions on a frame by frame basis. CSI configuration allows interleaving of DSI data frames from the parallel input signals with SPI

commands and data from the TX FIFO. The data returned from the bus slave is either used to drive the parallel output signals or it is stored in the RX FIFO. The CSI configuration allows serialized data and configuration or diagnostic data to be transferred to a slave device using only one serial link. The DSPI is in CSI configuration when the DCONF field in the DSPI_MCR is 0b10. Figure 30-23 shows an example of how a DSPI can be used with a deserializing peripheral that supports SPI control for control and diagnostic frames.

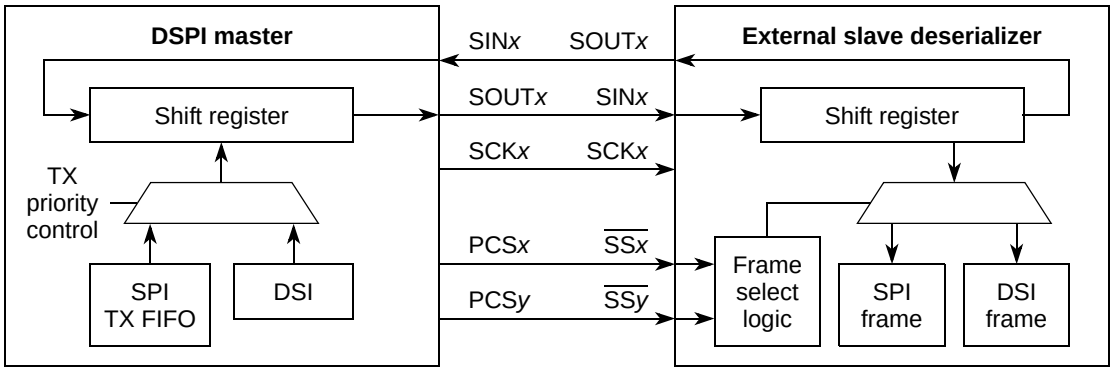


Figure 30-23. Example of System using DSPI in CSI Configuration

In CSI configuration, the DSPI transfers DSI data based on DSI Transfer Initiation Control. (See Section 30.4.4.5, DSI Transfer Initiation Control.) When there are SPI commands in the TX FIFO, the SPI data has priority over the DSI frames. When the TX FIFO is empty, DSI transfer resumes.

Two peripheral chip select signals indicate whether DSI data or SPI data is transmitted. The user must configure the DSPI so that the two CTAR registers associated with DSI data and SPI data assert different peripheral chip select signals denoted in the figure as PCS*x* and PCS*y*. The CSI configuration is only supported in master mode.

Data returned from the external slave while a DSI frame is transferred is placed on the parallel output signals. Data returned from the external slave while an SPI frame is transferred is moved to the RX FIFO. The TX FIFO and RX FIFO are fully functional in CSI mode.

30.4.5.1 CSI Serialization

Serialization in the CSI configuration is similar to serialization in DSI configuration. The transfer attributes for SPI frames are determined by the DSPI_CTAR*n* register selected by the CTAS field in the SPI command halfword. The transfer attributes for the DSI frames are determined by the DSPI_CTAR*n* register selected by the DSICTAS field in the DSPI_DSICR. Figure 30-24 shows the CSI serialization logic.

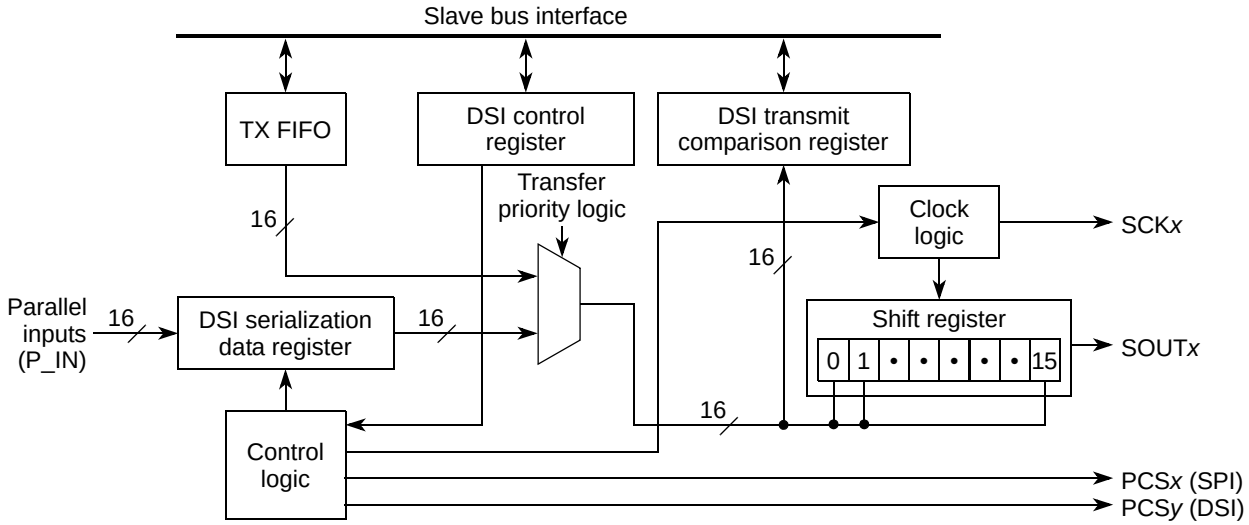


Figure 30-24. CSI Serialization Diagram

The parallel inputs signal states are latched into the DSPI DSI Serialization Data Register (DSPI_SDR) on the rising edge of every system clock and serialized based on the transfer initiation control settings in the DSPI_DSICR. For more information on the DSPI_SDR, refer to Section 30.3.2.11, DSPI DSI Serialization Data Register (DSPI_SDR). SPI frames written to the TX FIFO have priority over DSI data from the DSPI_SDR and are transferred at the next frame boundary. A copy of the most recently transferred DSI frame is stored in the DSPI_COMPR. The transfer priority logic selects the source of the serialized data and asserts the appropriate chip select signal.

30.4.5.2 CSI Deserialization

The deserialized frames in CSI configuration go into the DSPI_SDR or the RX FIFO based on the transfer priority logic. When DSI frames are transferred, the returned frames are deserialized and latched into the DSPI_DDR. When SPI frames are transferred, the returned frames are deserialized and written to the RX FIFO. Figure 30-25 shows the CSI deserialization logic.

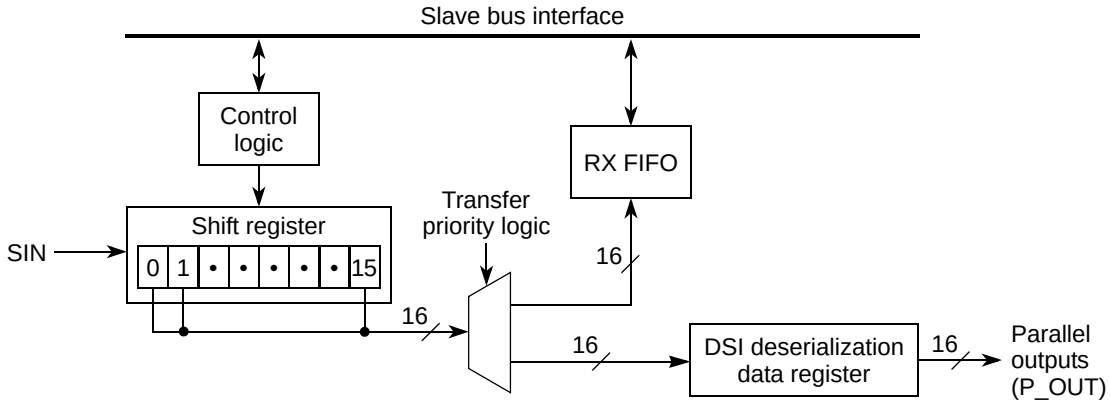


Figure 30-25. CSI Deserialization Diagram

30.4.6 Buffered SPI Operation

The DSPI can use a FIFO buffering mechanism to transmit and receive commands and data to and from external devices. The transmit FIFO buffers SPI commands and data to be transferred. The receive FIFO buffers incoming serial data. Both FIFOs are four entries deep. The TX FIFO stores 32-bit words when the DSPIs are configured for master mode. The 32-bit words are composed of 16-bit command fields and data fields as wide as 16 bits. The RX FIFOs store 16-bit words of received data from external devices. When the DSPI is configured for slave mode, the DSPI ignores the SPI command in the TX FIFO.

For queued operations, the SPI queues reside in system memory external to the DSPI. Data transfers between the memory and the DSPI FIFOs are accomplished through the use of the eDMA controller or through host software. See Figure 30-26 for conceptual diagram of the queue data transfer control in the MCU.

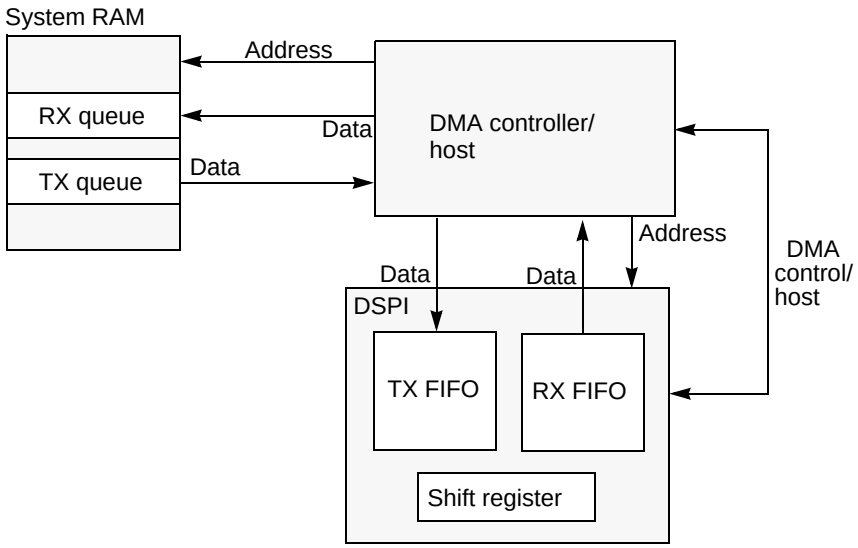


Figure 30-26. DSPI Queue Transfer Control in PXN20

30.4.7 DSPI Baud Rate and Clock Delay Generation

The SCK frequency and the delay values for serial transfer are generated by dividing the system clock frequency by a prescaler and a scaler with the option for doubling the baud rate. Figure 30-27 shows conceptually how the SCK signal is generated.

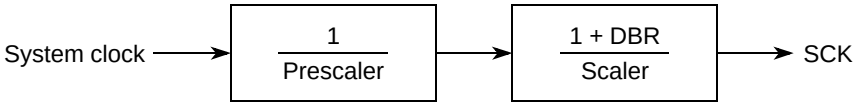


Figure 30-27. Communications Clock Prescalers and Scalers

30.4.7.1 Baud Rate Generator

The baud rate is the frequency of the serial communication clock (SCK). The system clock is divided by a baud rate prescaler (defined by DSPI_CTAR*n*[PBR]) and baud rate scaler (defined by

DSPI_CTAR*n*[BR]) to produce SCK with the possibility of halving the scaler division. The DBR, PBR, and BR fields in the DSPI_CTAR*n* registers select the frequency of SCK using the following formula:

Eqn. 30-5

$$\text{SCK baud rate} = \frac{f_{SYS}}{\text{PBRPrescalerValue}} \times \frac{1 + \text{DBR}}{\text{BRScalerValue}}$$

Table 30-26 shows an example of a computed baud rate.

Table 30-26. Baud Rate Computation Example

f_{SYS}	PBR	Prescaler Value	BR	Scaler Value	DBR Value	Baud Rate
66 MHz	0b00	2	0b0000	2	0	16.67 Mbit/s
20 MHz	0b00	2	0b0000	2	1	10 Mbit/s

30.4.7.2 PCS to SCK Delay (t_{CSC})

The PCS to SCK delay is the length of time from assertion of the PCS signal to the first SCK edge. See Figure 30-29 for an illustration of the PCS to SCK delay. The PCSSCK and CSSCK fields in the DSPI_CTAR*n* registers select the PCS to SCK delay, and the relationship is expressed by the following formula:

Eqn. 30-6

$$t_{CSC} = \frac{1}{f_{SYS}} \times \text{PCSSCK} \times \text{CSSCK}$$

Table 30-27 shows an example of the computed PCS to SCK delay.

Table 30-27. PCS to SCK Delay Computation Example

PCSSCK	Prescaler Value	CSSCK	Scaler Value	f_{SYS}	PCS to SCK Delay
0b01	3	0b0100	32	100 MHz	0.96 µs

30.4.7.3 After SCK Delay (t_{ASC})

The after SCK delay is the length of time between the last edge of SCK and the negation of PCS. See Figure 30-29 and Figure 30-30 for illustrations of the after SCK delay. The PASC and ASC fields in the DSPI_CTAR*n* registers select the after SCK delay. The relationship between these variables is given in the following formula:

$$t_{ASC} = \frac{1}{f_{SYS}} \times \text{PASC} \times \text{ASC}$$

Table 30-28 shows an example of the computed after SCK delay.

Table 30-28. After SCK Delay Computation Example

PASC	Prescaler Value	ASC	Scaler Value	Fsys	After SCK Delay
0b01	3	0b0100	32	100 MHz	0.96 us

30.4.7.4 Delay after Transfer (t_{DT})

The delay after transfer is the length of time between negation of the PCS signal for a frame and the assertion of the PCS signal for the next frame. See Figure 30-29 for an illustration of the delay after transfer. The PDT and DT fields in the DSPI_CTAR*n* registers select the delay after transfer. The following formula expresses the PDT/DT/delay after transfer relationship:

Eqn. 30-7

$$t_{DT} = \frac{1}{f_{SYS}} \times PDT \times DT$$

Table 30-29 shows an example of the computed delay after transfer.

Table 30-29. Delay after Transfer Computation Example

PDT	Prescaler Value	DT	Scaler Value	f_{SYS}	Delay after Transfer
0b01	3	0b1110	32768	100 MHz	0.98 ms

When in non-continuous clock mode the T_{DT} delay is configurable as outlined in the DSPI_CTAR*n* registers. When in continuous clock mode and TSB is not enabled the delay is fixed at 1 SCK period. When in TSB and continuous mode, the delay is programmed as outlined in the DSPI_CTAR*n* registers. In event that the delay does not coincide with an SCK period in duration, the delay is extended to the next SCK active edge. Table 30-30 shows an example of how to compute the delay after Transfer with the clock period of SCK defined as T_{SCK}. The values calculated assume 1 TSCK period = 4 ipg_clk.

Table 30-30. Delay after Transfer Computation Example in TSB Configuration

DT field	Tdt[1] (Tsck)	PDT field 0	1	2	3
	0[2]	1	2	3	4
	1	1	3	5	7
	2	2	6	10	14
	3	4	12	20	28
	4	8	24	40	56
	5	16	48	80	112
	6	32	96	160	224
	7	64	192	320	448
	8	128	384	640	896
	9	256	768	1280	1792
	10	512	1536	2560	3584
	11	1024	3072	5120	7168
	12	2048	6144	10240	14336
	13	4096	12288	20480	28672
	14	8192	24576	40960	57344
	15	16384	49152	81920	114688

1 Some values are not reachable (e.g., 9, 11, 13, 15, 17, 18, 19...), to calculate these values, please see the Equation 30-3.

2 The values in this row were rounded to the next integer value.

30.4.7.5 Peripheral Chip Select Strobe Enable ($\overline{\text{PCSS}}$)

The $\overline{\text{PCSS}}$ signal provides a delay to allow the PCS signals to settle after a transition occurs, thereby avoiding glitches. When the DSPI is in master mode and PCSSE bit is set in the DSPI_MCR, $\overline{\text{PCSS}}$ provides a signal for an external demultiplexer to decode the PCS[0:4] signals into as many as 32 glitch-free PCS signals. Figure 30-28 shows the timing of the $\overline{\text{PCSS}}$ signal relative to PCS signals.

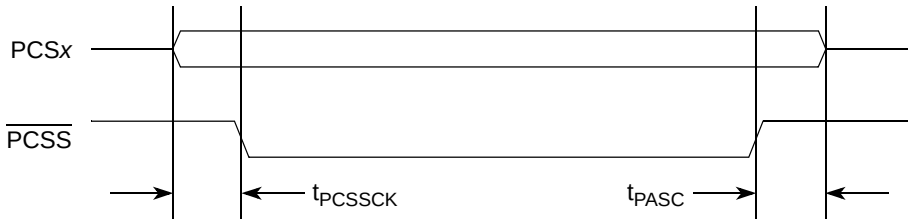


Figure 30-28. Peripheral Chip Select Strobe Timing

The delay between the assertion of the PCS*x* signals and the assertion of $\overline{\text{PCSS}}$ is selected by the PCSSCK field in the DSPI_CTAR*n* register based on the following formula:

Eqn. 30-8

$$tPCSSCK = \frac{1}{f_{SYS}} \times PCSSCK$$

At the end of the transfer the delay between $\overline{\text{PCSS}}$ negation and PCS*x* negation is selected by the PASC field in the DSPI_CTAR*n* register based on the following formula:

Eqn. 30-9

$$tPASC = \frac{1}{f_{SYS}} \times PASC$$

Table 30-31 shows an example of the computed t_{PCSSCK} delay.

Table 30-31. Peripheral Chip Select Strobe Assert Computation Example

PCSSCK	Prescaler	f_{SYS}	Delay before Transfer
0b11	7	100 MHz	70.0 ns

Table 30-32 shows an example of the computed the t_{PASC} delay.

Table 30-32. Peripheral Chip Select Strobe Negate Computation Example

PASC	Prescaler	f_{SYS}	Delay after Transfer
0b11	7	100 MHz	70.0 ns

The $\overline{\text{PCSS}}$ signal is not supported when Continuous Serial Communication SCK is enabled (CONT_SCKE = 1).

30.4.8 Transfer Formats

The SPI serial communication is controlled by the serial communications clock (SCK) signal and the PCS signals. The SCK signal provided by the master device synchronizes shifting and sampling of the data on the SIN and SOUT pins. The PCS signals serve as enable signals for the slave devices.

When the DSPI is the bus master, the CPOL and CPHA bits in the DSPI clock and transfer attributes registers (DSPI_CTAR*n*) select the polarity and phase of the serial clock, SCK. The polarity bit selects the idle state of the SCK. The clock phase bit selects if the data on SOUT is valid before or on the first SCK edge.

When the DSPI is the bus slave, CPOL and CPHA bits in the DSPI_CTAR0 (SPI) or DSPI_CTAR1 (DSI) select the polarity and phase of the serial clock. Even though the bus Slave does not control the SCK signal, clock polarity, clock phase and number of bits to transfer must be identical for the master device and the slave device to ensure proper transmission.

The DSPI supports four different transfer formats:

- Classic SPI with CPHA = 0
- Classic SPI with CPHA = 1
- Modified transfer format with CPHA = 0
- Modified transfer format with CPHA = 1

A modified transfer format is supported to allow for high-speed communication with peripherals that require longer setup times. The DSPI can sample the incoming data later than halfway through the cycle to give the peripheral more setup time. The MTFE bit in the DSPI_MCR selects between Classic SPI Format and Modified Transfer Format. The Classic SPI Formats are described in Section 30.4.8.1, Classic SPI Transfer Format (CPHA = 0), and Section 30.4.8.2, Classic SPI Transfer Format (CPHA = 1). The Modified Transfer Formats are described in Section 30.4.8.3, Modified SPI/DSI Transfer Format (MTFE = 1, CPHA = 0), and Section 30.4.8.4, Modified SPI/DSI Transfer Format (MTFE = 1, CPHA = 1).

In the SPI and DSI configurations, the DSPI provides the option of keeping the PCS signals asserted between frames. See Section 30.4.8.5, Continuous Selection Format, for details.

30.4.8.1 Classic SPI Transfer Format (CPHA = 0)

The transfer format shown in Figure 30-29 is used to communicate with peripheral SPI slave devices where the first data bit is available on the first clock edge. In this format, the master and slave sample their SIN pins on the odd-numbered SCK edges and change the data on their SOUT pins on the even-numbered SCK edges.

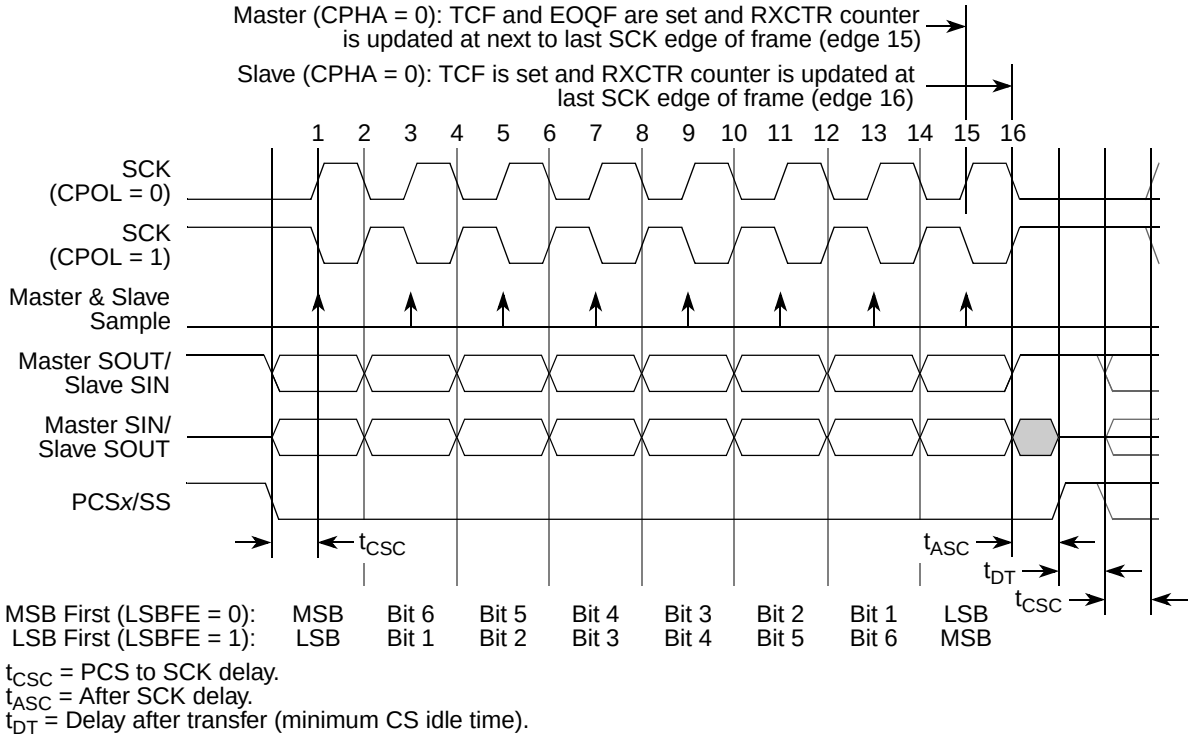


Figure 30-29. DSPI Transfer Timing Diagram (MTFE = 0, CPHA = 0, FMSZ = 8)

The master initiates the transfer by placing its first data bit on the SOUT*x* pin and asserting the appropriate peripheral chip select signals to the slave device. The slave responds by placing its first data bit on its SOUT pin. After the t_{CSC} delay has elapsed, the master outputs the first edge of SCK. This is the edge used by the master and slave devices to sample the first input data bit on their serial data input signals. At the second edge of the SCK, the master and slave devices place their second data bit on their serial data output signals. For the rest of the frame the master and the slave sample their SIN pins on the odd-numbered clock edges and changes the data on their SOUT pins on the even-numbered clock edges. After the last clock edge occurs a delay of t_{ASC} is inserted before the master negates the PCS signals. A delay of t_{DT} is inserted before a new frame transfer can be initiated by the master.

For the CPHA = 0 condition of the master, TCF and EOQF are set and the RXCTR counter is updated at the next to last serial clock edge of the frame (edge 15) of Figure 30-29.

For the CPHA = 0 condition of the slave, TCF is set and the RXCTR counter is updated at the last serial clock edge of the frame (edge 16) of Figure 30-29.

30.4.8.2 Classic SPI Transfer Format (CPHA = 1)

This transfer format shown in Figure 30-30 is used to communicate with peripheral SPI slave devices that require the first SCK edge before the first data bit becomes available on the slave SOUT pin. In this format the master and slave devices change the data on their SOUT pins on the odd-numbered SCK edges and sample the data on their SIN pins on the even-numbered SCK edges

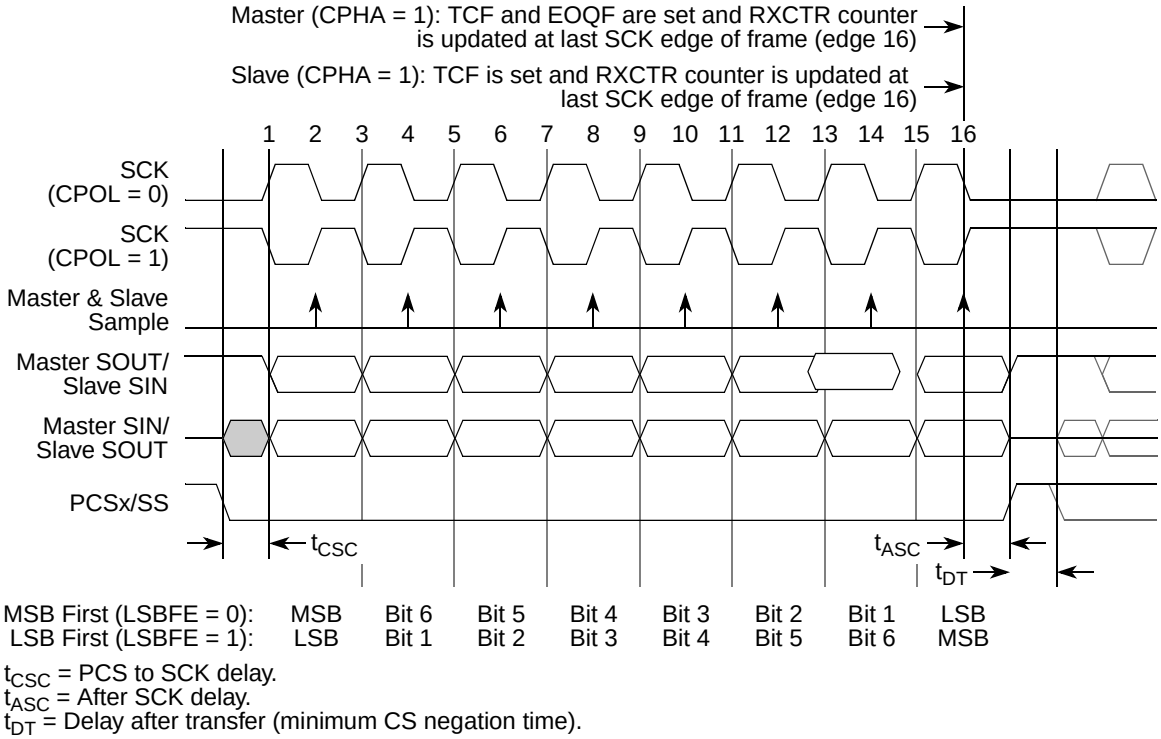


Figure 30-30. DSPI Transfer Timing Diagram (MTFE = 0, CPHA = 1, FMSZ = 8)

The master initiates the transfer by asserting the PCS signal to the slave. After the t_{CSC} delay has elapsed, the master generates the first SCK edge and at the same time places valid data on the master SOUT pin. The slave responds to the first SCK edge by placing its first data bit on its slave SOUT pin.

At the second edge of the SCK the master and slave sample their SIN pins. For the rest of the frame the master and the slave change the data on their SOUT pins on the odd-numbered clock edges and sample their SIN pins on the even-numbered clock edges. After the last clock edge occurs a delay of t_{ASC} is inserted before the master negates the PCS signal. A delay of t_{DT} is inserted before a new frame transfer can be initiated by the master.

For CPHA = 1 the master EOQF and TCF and slave TCF are set at the last serial clock edge (edge 16) of Figure 30-30. For CPHA = 1 the master and slave RXCTR counters are updated on the same clock edge.

30.4.8.3 Modified SPI/DSI Transfer Format (MTFE = 1, CPHA = 0)

In this modified transfer format, both the master and the slave sample later in the SCK period than in classic SPI mode to allow for delays in device pads and board traces. These delays become a more significant fraction of the SCK period as the SCK period decreases with increasing baud rates.

NOTE

For correct operation of the modified transfer format, the user must thoroughly analyze the SPI link timing budget.

The master and the slave place data on the SOUT pins at the assertion of the PCS signal. After the PCS to SCK delay has elapsed, the first SCK edge is generated. The slave samples the master SOUT signal on

every odd-numbered SCK edge. The slave also places new data on the slave SOUT on every odd-numbered clock edge.

The master places its second data bit on the SOUT line one system clock after odd-numbered SCK edge. The point where the master samples the slave SOUT is selected by writing to the SMPL_PT field in the DSPI_MCR. Table 30-33 lists the number of system clock cycles between the active edge of SCK and the master sample point. The master sample point can be delayed by one or two system clock cycles.

Table 30-33. Delayed Master Sample Point

SMPL_PT	Number of System Clock Cycles Between Odd-Numbered Edge of SCK and Sampling of SIN
00	0
01	1
10	2
11	Reserved

Figure 30-31 shows the modified transfer format for CPHA = 0. Only the condition where CPOL = 0 is illustrated. The delayed master sample points are indicated with a lighter shaded arrow.

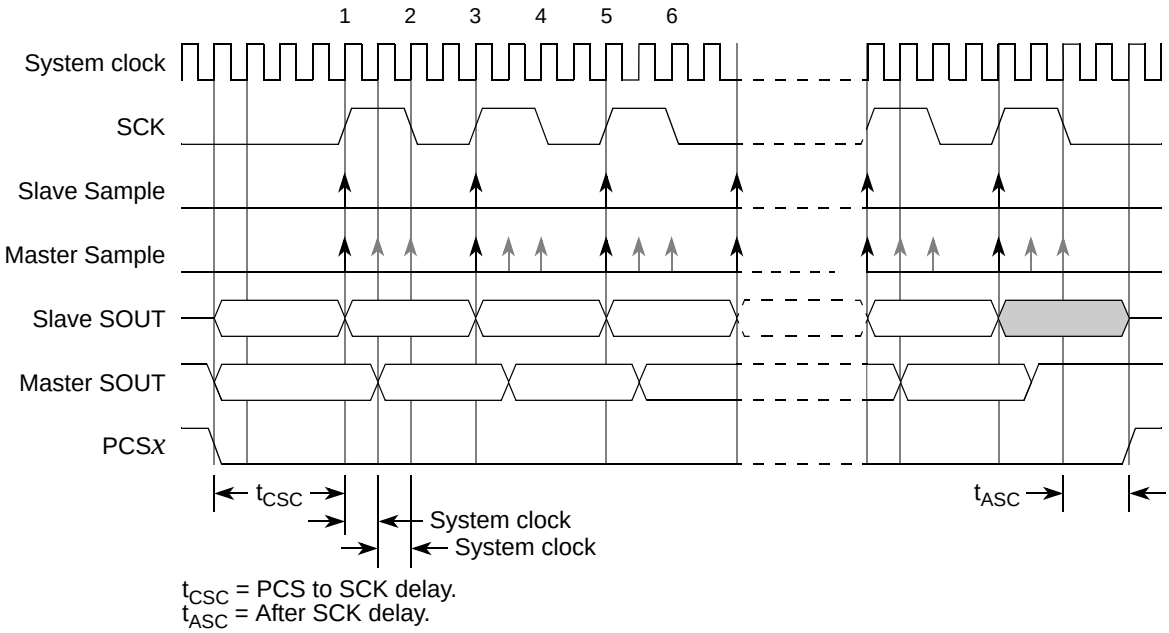


Figure 30-31. DSPI Modified Transfer Format (MTFE = 1, CPHA = 0, Fsck = Fsys/4)

30.4.8.4 Modified SPI/DSI Transfer Format (MTFE = 1, CPHA = 1)

Figure 30-32 shows the modified transfer format for CPHA = 1. Only the condition where CPOL = 0 is described. At the start of a transfer, the DSPI asserts the PCS signal to the slave device. After the PCS to SCK delay has elapsed, the master and the slave put data on their SOUT pins at the first edge of SCK. The slave samples the master SOUT signal on the even-numbered edges of SCK. The master samples the slave SOUT signal on the odd-numbered SCK edges starting with the third SCK edge. The slave samples the

last bit on the last edge of the SCK. The master samples the last slave SOUT bit one half SCK cycle after the last edge of SCK. No clock edge is visible on the master SCK pin during the sampling of the last bit. The SCK to PCS delay must be greater or equal to half of the SCK period.

NOTE

For correct operation of the modified transfer format, the user must thoroughly analyze the SPI link timing budget.

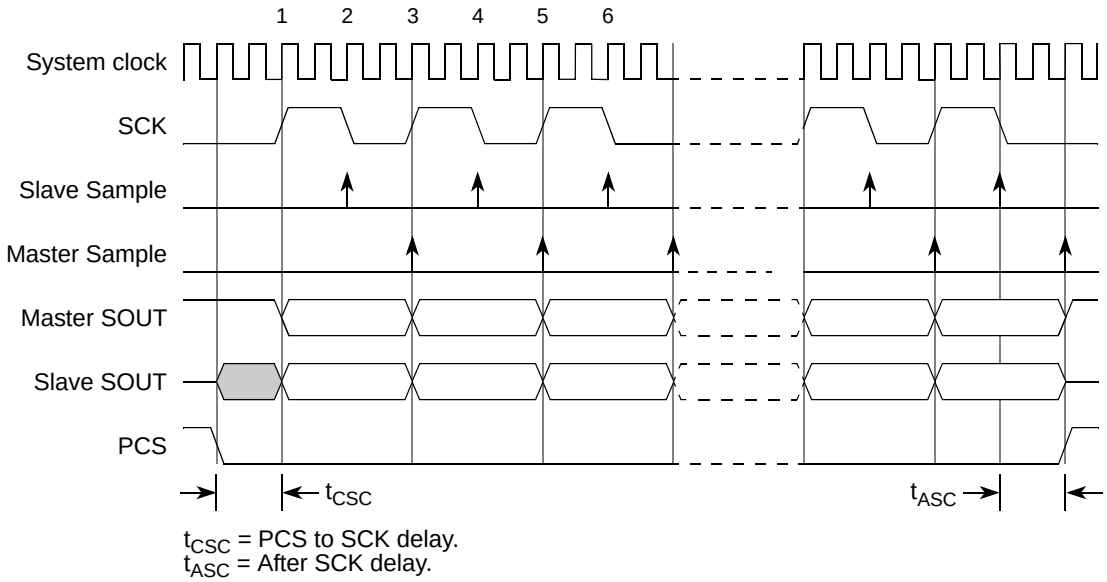


Figure 30-32. DSPI Modified Transfer Format (MTFE = 1, CPHA = 1, Fsck = Fsys/4)

NOTE

When the DSPI is being used in the Modified Transfer Format mode (DSPI_MCR[MTFE]=1) with the clock phase set for data changing on the leading edge of the clock and captured on the following edge in the DSPI Clock and Transfer Attributes Register (DSPI_CTARn[CPHA]=1), if the After SCK delay scaler (ASC) time is set to less than 1/2 SCK clock period the DSPI may not complete the transaction - the TCF flag will not be set, serial data will not received, and last transmitted bit can be truncated.

In this case, the Modified Transfer Format mode is required DSPI_MCR[MTFE]=1 with the clock phase set for serial data changing on the leading edge of the clock and captured on the following edge in the SCK clock (Transfer Attributes Register (DSPI_CTARn[CPHA]=1) make sure that the ASC time is set to be longer than half SCK clock period.

30.4.8.5 Continuous Selection Format

Some peripherals must be deselected between every transfer. Other peripherals must remain selected between several sequential serial transfers. The continuous selection format provides the flexibility to handle both cases. The continuous selection format is enabled for the SPI configuration by setting the

CONT bit in the SPI command. Continuous selection is enabled for the DSI configuration by setting the DCONT bit in the DSPI_DSICR. The behavior of the PCS signals in the two configurations is identical, so only SPI configuration is described.

When the CONT bit = 0, the DSPI drives the asserted chip select signals to their idle states in between frames. The idle states of the chip select signals are selected by the PCSIS field in the DSPI_MCR. Figure 30-33 shows the timing diagram for two 4-bit transfers with CPHA = 1 and CONT = 0.

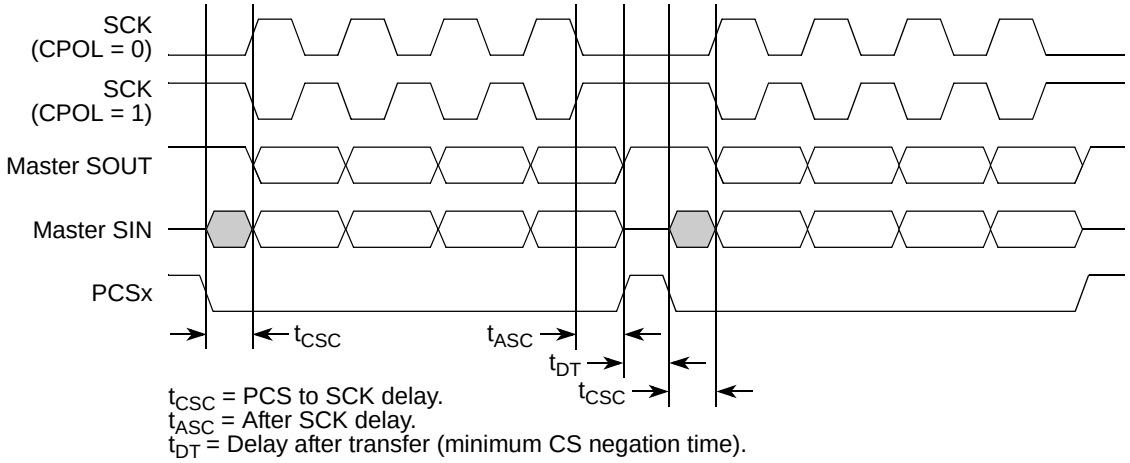


Figure 30-33. Example of Non-Continuous Format (CPHA = 1, CONT = 0)

When the CONT bit = 1and the PCS signal for the next transfer is the same as for the current transfer, the PCS signal remains asserted for the duration of the two transfers. The delay between transfers (t_{DT}) is not inserted between the transfers. Figure 30-34 shows the timing diagram for two 4-bit transfers with CPHA = 1 and CONT = 1.

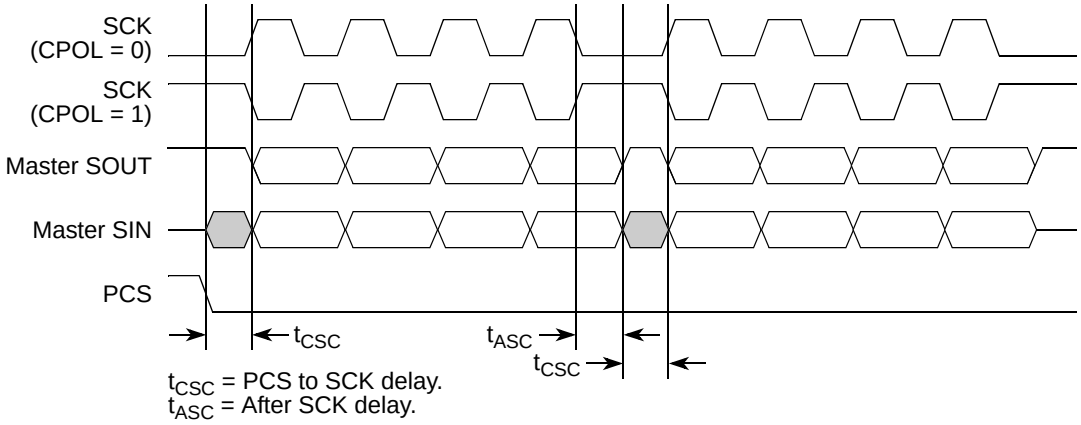


Figure 30-34. Example of Continuous Transfer (CPHA = 1, CONT = 1)

In Figure 30-34, the period length at the start of the next transfer is the sum of t_{ASC} and t_{CSC}; i.e., it does not include a half-clock period. The default settings for these provide a total of four system clocks. In many situations, t_{ASC} and t_{CSC} must be increased if a full half-clock period is required.

Switching CTAR registers or changing which PCS signals are asserted between frames while using continuous selection can cause errors in the transfer. The PCS signal should be negated before CTAR is switched or different PCS signals are selected.

30.4.8.6 Clock Polarity Switching Between DSPI Transfers

If it is desired to switch polarity between non-continuous DSPI frames, the edge generated by the change in the idle state of the clock occurs one system clock before the assertion of the chip select for the next frame. In Figure 30-35, time A shows the one clock interval. Time B is user programmable from a minimum of two system clocks. Refer to Section 30.3.2.3, DSPI Clock and Transfer Attributes Registers 0–7 (DSPI_CTARn).

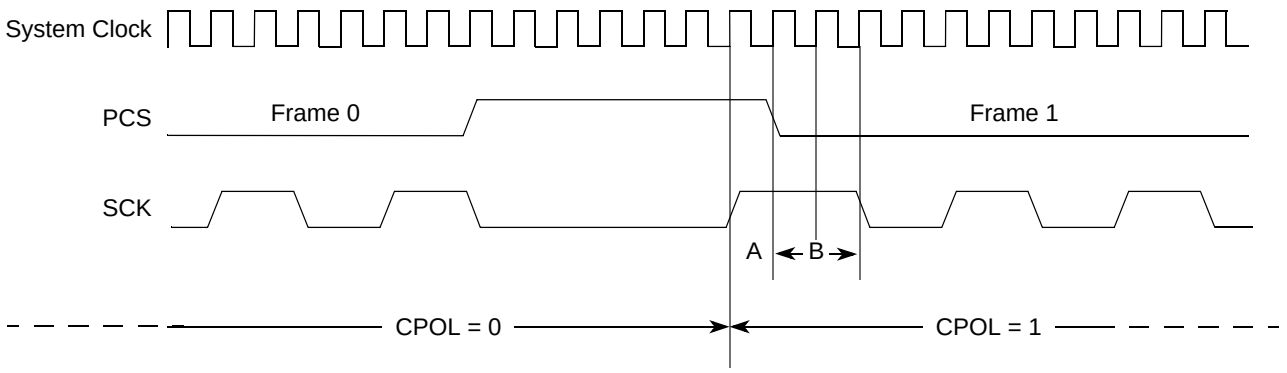


Figure 30-35. Polarity Switching Between Frames

30.4.9 Continuous Serial Communications Clock

The DSPI provides the option of generating a continuous SCK signal for slave peripherals that require a continuous clock.

Continuous SCK is enabled by setting the CONT_SCKE bit in the DSPI_MCR. Continuous SCK is valid in all configurations.

Continuous SCK is only supported for CPHA = 1. Setting CPHA = 0 is ignored if the CONT_SCKE bit is set. Continuous SCK is supported for modified transfer format.

- Clock and transfer attributes for the continuous SCK mode are set according to the following rules: When the DSPI is in SPI configuration, CTAR0 shall be used initially. At the start of each SPI frame transfer, the CTAR specified by the CTAS for the frame shall be used.
- When the DSPI is in DSI configuration, the CTAR specified by the DSICTAS field shall be used at all times.
- When the DSPI is in CSI configuration, the CTAR selected by the DSICTAS field shall be used initially. At the start of a SPI frame transfer, the CTAR specified by the CTAS value for the frame shall be used. At the start of a DSI frame transfer, the CTAR specified by the DSICTAS field shall be used.
- In all configurations, the currently selected CTAR shall remain in use until the start of a frame with a different CTAR specified, or the continuous SCK mode is terminated.

It is recommended that the baud rate is the same for all transfers made while using the continuous SCK. Switching clock polarity between frames while using continuous SCK can cause errors in the transfer. Continuous SCK operation is not guaranteed if the DSPI is put into halt mode or module disable mode.

Enabling continuous SCK disables the PCS to SCK delay and the delay after transfer (T_{DT}) is fixed at one T_{SCK} cycle. When TSB configuration is enabled the T_{DT} is programmable to a minimum of $1xT_{SCK}$ cycles by configuring PDT and DT values in the respective CTAR register. Figure 30-36 shows timing diagram for continuous SCK format with continuous selection disabled.

Enabling continuous SCK disables the PCS to SCK delay and the after SCK delay. The delay after transfer is fixed at one SCK cycle. Figure 30-36 shows timing diagram for continuous SCK format with continuous selection disabled.

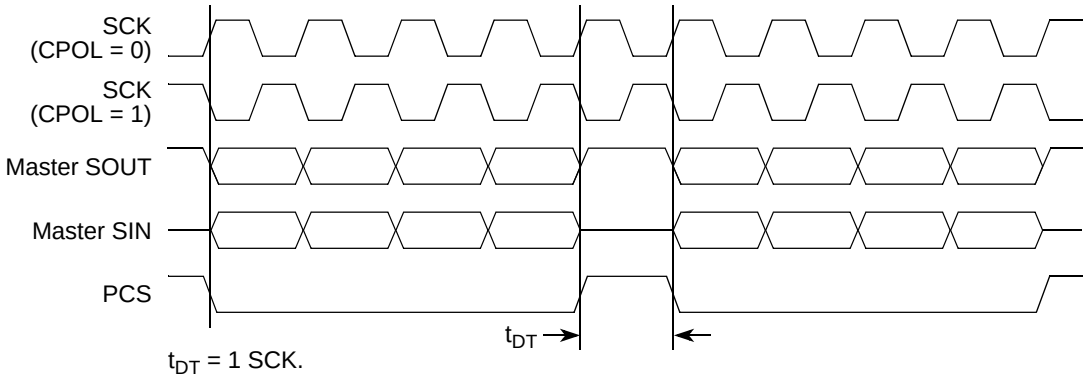


Figure 30-36. Continuous SCK Timing Diagram (CONT = 0)

If the CONT bit in the TX FIFO entry is set or the DCONT in the DSPI_DSICR is set, PCS remains asserted between the transfers. Under certain conditions, SCK can continue with PCS asserted, but with no data being shifted out of SOUT (SOUT pulled high). This can cause the slave to receive incorrect data. Those conditions include:

- Continuous SCK with CONT bit set, but no data in the transmit FIFO.
- Continuous SCK with CONT bit set and entering STOPPED state (refer to Section 30.4.2, Start and Stop of DSPI Transfers).
- Continuous SCK with CONT bit set and entering halt mode or module disable mode.

Figure 30-37 shows timing diagram for continuous SCK format with continuous selection enabled.

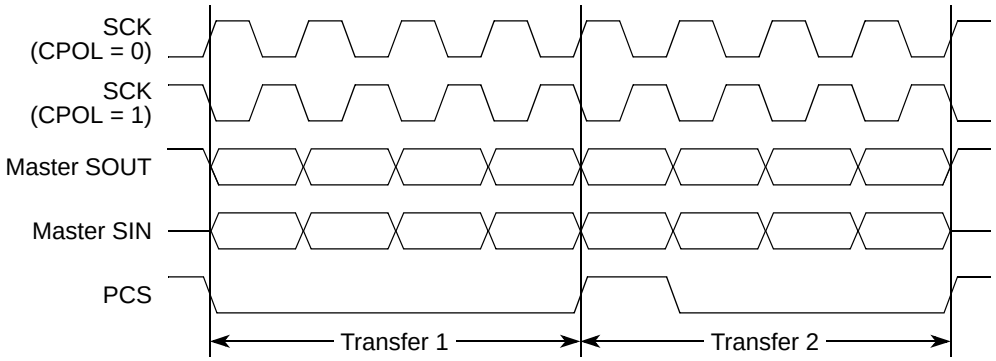


Figure 30-37. Continuous SCK Timing Diagram (CONT = 1)

30.4.10 Timed Serial Bus (TSB)

The DSPI can be programmed in timed serial bus (TSB) configuration by asserting the TSBC bit in the DSPI_DSICR register. See Section 30.3.2.10, DSPI DSI Configuration Register (DSPI_DSICR), for details. To operate in TSB configuration, the DSPI must be in master mode and configured as DSI (DCONF = 0b01). The TSB allows operating in continuous and non-continuous serial communication clock (controlled by bit CONT_SCKE).

Figure 30-38 shows the signals used in the TSB interface. The SDR and ASDR registers are set to 32 bits in this configuration, to allow the Micro Second Channel (MSC) feature to be performed.

In the TSB configuration, the DSPI can send from 4 to as many as 32 data bits. The source of these bits can be either the DSPI DSI Alternate Serialization Data Register (DSPI_ASDR), written by the host software, or the parallel input pin states latched into the DSPI DSI Serialization Data Register (DSPI_SDR).

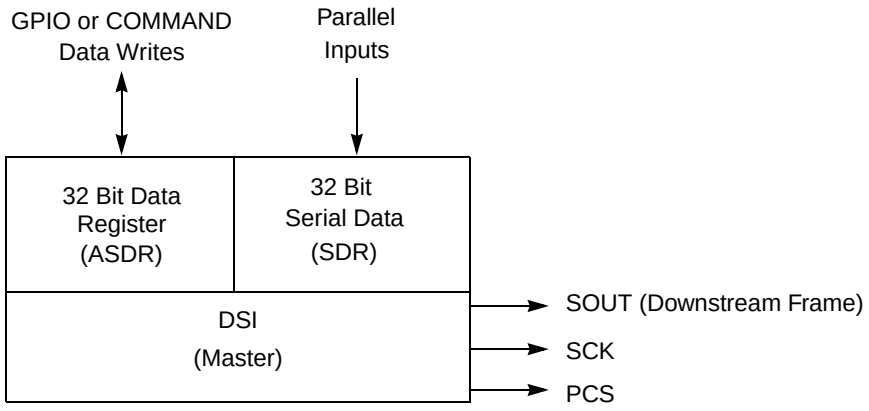


Figure 30-38. DSPI Usage in the TSB Configuration

The same constraints that apply to DSI are valid for TSB except for the frame size and the Delay After Transfer value (T_{DT}). The TSB configuration allows from 4 to 32 bits frame size to be used, and T_{DT} can be programmable to a minimum of $1 \times T_{SCK}$, allowing a programmable inter-message gap. See Table 30-10 and Table 30-30 for details on programming the T_{DT} values.

The time between the negation of the CS at the end of one frame to the assertion of CS at the next frame is defined by: $T_{DT} = P_{DT} \times DT / F_{sys,}$ but delayed until the next active edge of T_{SCK}. The gap is only be whole period of TSCK and the PDT and DT fields of the specific DSPI_CTAR*n* register select the delay after transfer. Some values are possible.

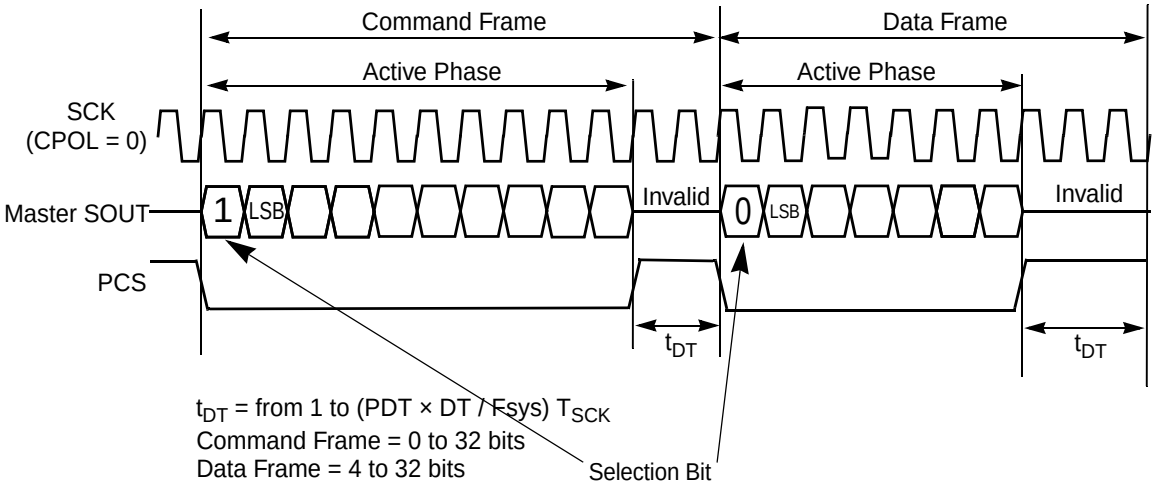


Figure 30-39. TSB Downstream Frame

Figure 30-39 shows the two types of downstream frames used in the TSB configuration, command frame and data frame. Refer to Section 30.4.10.1, PCS Switch Over Timing, and Section 30.4.10.3, TSB Data Frame Format, for detailed information. The Command Word can be written by software, and the Data Word consist of 32-bit words, from the SDR or ASDR registers. Only the downstream frame is supported in the TSB configuration. The upstream frame can be handled by software using any available serial input.

The selection bit, the start bit for a frame, is not a requirement but could be implemented by software. The number of frame bits can be in the range of 4 to 32 bits. In this configuration, the least significant bit of a frame should be transmitted first (LSBFE = 1).

30.4.10.1 PCS Switch Over Timing

When in TSB mode, it is possible to switch the set of PCS signals that are driven during the first part of the frame to a different set of PCS signals during the second part of the frame. The bit at which this switch over occurs is contained in the DSICR register.

In order to maximize both the setup and hold time margins on the old and new PCS signals, the timing of the switch over occurs on the active edge of the master SCK data capture between the last bit of the first part of the frame and the first bit of the second part of the frame. For example, if the first part of the frame is 5 bits and the second part of the frame is 10 bits, the PCS signals switch at the active edge of the master SCK in between bits 5 and 6 of the frame as seen by the slave. The exact timing between the external signals SCK and PCS signals is not exactly aligned due to routing and pad differences. This approach ensures that larger/shorter SCK periods result in approximately symmetric increases/decreases of setup and hold margins on the PCS signals. The setup time for the PCS signals before the first bit of the first part of the frame and the hold time for the PCS signals after the last bit of the second part of the frame are

unchanged in this mode with respect to the other modes and remain controlled by the CSC and ASC delay fields respectively when not in continuous SCK mode.

30.4.10.2 TSB Command Frame Format

In the TSB configuration a command frame is shown in Figure 30-40.

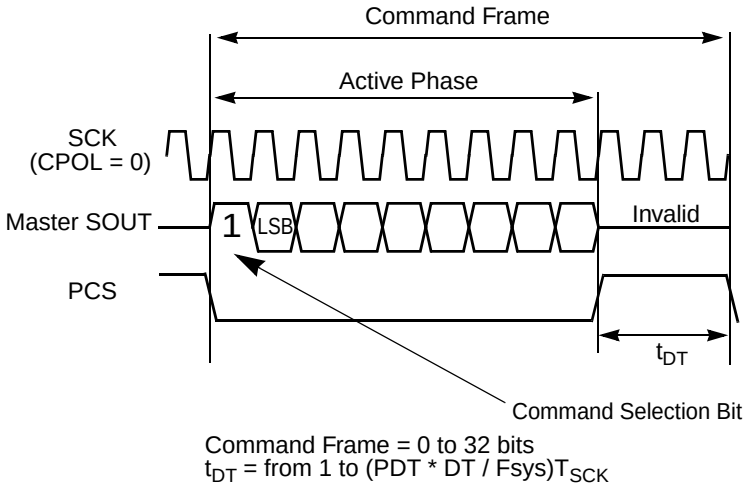


Figure 30-40. Command Frame Format

In the active phase of the command frame, the chip select becomes active validating the SOUT and SCK output signals. Outside the active phase, chip select is at passive level and invalid data may occur at SOUT.

For TSB configuration, assuming CPOL = 0, the SOUT output and the chip select changes its state always with the SCK rising edge. The SOUT signal must be sampled in the slave device with the falling edge of SCK. The clock period of SCK is defined as T_{SCK}. The length of the command frame passive phase T_{DT} should always be fixed to a minimum of 1x T_{SCK}.

30.4.10.3 TSB Data Frame Format

A data frame is transmitted from the TSB controller to the receiving devices. Figure 30-41 details the frame active and passive phase,

For TSB configuration assuming CPOL = 0, the SOUT output and the chip select change their states always with the SCK rising edge. The SOUT signal must be sampled in the receiving device with the falling edge of SCK. The length of the data frame passive phase T_{DT} can be a minimum of $1 \times T_{SCK}$.

A data frame can be composed by data bits only or by data bits preceded by a selection bit, see Figure 30-41. A data frame with a selection bit always starts with a low level bit at SOUT.

The number of data bits in the active phase is from 4 to 32 bits, and the least significant bit of a data portion is transmitted first (LSBFE = 1).

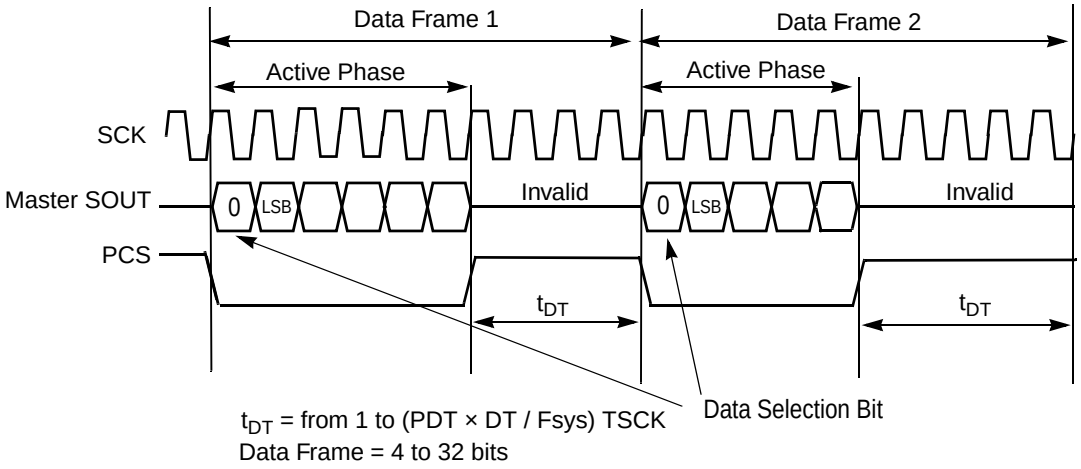


Figure 30-41. TSB Data Frame Format

30.4.11 Peripheral Chip Select Expansion and Deglitching

The DSPI supports as many as 64 peripheral chip select signals with the use of an external demultiplexer. As many as 32 peripheral chip select signals can be used if deglitching is desired. The $\overline{\text{PCSS}}$ signal provides the appropriate timing to enable and disable the demultiplexer for the PCS[0:4] signals.

Figure 30-42 shows how an external 5-to-32 demultiplexer (decoder) can be connected to the DSPI.

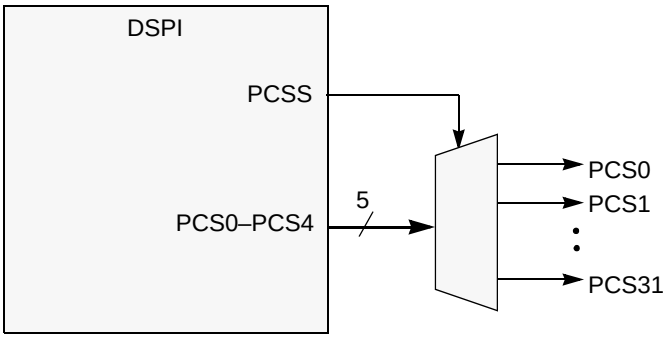


Figure 30-42. DSPI PCS Expansion and Deglitching

30.4.12 DMA and Interrupt Conditions

The DSPI has six conditions that can generate interrupt requests only and two conditions that can generate interrupt or DMA request. Table 30-34 lists the conditions. The **X** indicates which signals are connected.

Table 30-34. DSPI Interrupt and DMA Request Conditions

Condition	Flag	Interrupt	DMA
End of queue reached (EOQ)	EOQF	X	
TX FIFO is not full	TFFF	X	X
Transfer of current frame complete	TCF	X	

Table 30-34. DSPI Interrupt and DMA Request Conditions (continued)

Condition	Flag	Interrupt	DMA
Attempt to transmit with an empty transmit FIFO	TFUF	X	
RX FIFO is not empty	RFDF	X	X
Frame received while receive FIFO is full	RFOF	X	

Each condition has a flag bit in the DSPI Status Register (DSPI_SR) and an Request Enable bit in the DSPI DMA/Interrupt Request Select and Enable Register (DSPI_RSER). The TX FIFO Fill Flag (TFFF) and RX FIFO Drain Flag (RFDF) generate interrupt requests or DMA requests depending on the TFFF_DIRS and RFDF_DIRS bits in the DSPI_RSER. See Table 23-4 for the DSPI DMA channel assignments and Table 10-12 for the DSPI interrupt vectors.

30.4.12.1 End of Queue Interrupt Request (EOQF)

The end of queue request indicates that the end of a transmit queue is reached. The end of queue request is generated when the EOQ bit in the executing SPI command is asserted and the EOQF_RE bit in the DSPI_RSER is asserted. See the EOQ bit description in Section 30.3.2.4, DSPI Status Register (DSPI_SR). Refer to Figure 30-29 and Figure 30-30 that illustrate when EOQF is set.

30.4.12.2 Transmit FIFO Fill Interrupt or DMA Request (TFFF)

The transmit FIFO fill request indicates that the TX FIFO is not full. The transmit FIFO fill request is generated when the number of entries in the TX FIFO is less than the maximum number of possible entries, and the TFFF_RE bit in the DSPI_RSER is asserted. The TFFF_DIRS bit in the DSPI_RSER selects whether a DMA request or an interrupt request is generated.

30.4.12.3 Transfer Complete Interrupt Request (TCF)

The transfer complete request indicates the end of the transfer of a serial frame. The transfer complete request is generated at the end of each frame transfer when the TCF_RE bit is set in the DSPI_RSER. See the TCF bit description in Section 30.3.2.4, DSPI Status Register (DSPI_SR). Refer to Figure 30-29 and Figure 30-30 that illustrate when TCF is set.

30.4.12.4 Transmit FIFO Underflow Interrupt Request (TFUF)

The transmit FIFO underflow request indicates that an underflow condition in the TX FIFO has occurred. The transmit underflow condition is detected only for DSPI blocks operating in slave mode and SPI configuration. The TFUF bit is set when the TX FIFO of a DSPI operating in slave mode and SPI configuration is empty, and a transfer is initiated from an external SPI master. If the TFUF bit is set while the TFUF_RE bit in the DSPI_RSER is asserted, an interrupt request is generated.

30.4.12.5 Receive FIFO Drain Interrupt or DMA Request (RFDF)

The receive FIFO drain request indicates that the RX FIFO is not empty. The receive FIFO drain request is generated when the number of entries in the RX FIFO is not zero, and the RFDF_RE bit in the

DSPI_RSER is asserted. The RFDF_DIRS bit in the DSPI_RSER selects whether a DMA request or an interrupt request is generated.

30.4.12.6 Receive FIFO Overflow Interrupt Request

The receive FIFO overflow request indicates that an overflow condition in the RX FIFO has occurred. A receive FIFO overflow request is generated when RX FIFO and shift register are full and a transfer is initiated. The RFOF_RE bit in the DSPI_RSER must be set for the interrupt request to be generated.

Depending on the state of the ROOE bit in the DSPI_MCR, the data from the transfer that generated the overflow is either ignored or shifted in to the shift register. If the ROOE bit is set, the incoming data is shifted in to the shift register. If the ROOE bit is negated, the incoming data is ignored.

30.4.12.7 DMA Requests

The connection of the DSPI DMA request signals to the DMA channel mux is described in Table 23-4.

30.4.12.8 Interrupt Requests

The DSPI interrupts are connected as described in Table 23-4.

30.4.13 Power Saving Features

The DSPI supports three power-saving strategies:

- Halt mode
- Module disable mode—clock gating of non-memory mapped logic
- Clock gating of slave interface signals and clock to memory-mapped logic

30.4.13.1 Halt Mode

By setting the appropriate bit in the SIU_HLT0 register, a request is made to shut down all clocks in the DSPI. If no serial transfer is in progress, the DSPI immediately asserts an acknowledge signal to the system, allowing the clocks to be disabled. If a serial transfer is in progress when the request is received, the DSPI waits until it reaches a frame boundary before it asserts the acknowledge signal to the system. The status of this acknowledge signal can be determined by reading the SIU_HLTACK0 register.

While the clocks are shut off, the DSPI memory-mapped logic is not accessible. The states of the interrupt and DMA request signals cannot be changed while in halt mode.

Halt mode is exited by negating the appropriate bit in the SIU_HLT0 register.

30.4.13.2 Module Disable Mode

Module disable mode is a block-specific mode that the DSPI can enter to save power. Host software can initiate the module disable mode by writing a 1 to the MDIS bit in the DSPI_MCR.

When the MDIS bit is asserted, the DSPI negates ipg_enable_clk at the next frame boundary. If implemented, the ipg_enable_clk signal can stop the clock to the non-memory mapped logic. When

ipg_enable_clk is negated, the DSPI is in a dormant state, but the memory mapped registers are still accessible. Certain read or write operations have a different affect when the DSPI is in the module disable mode. Reading the RX FIFO pop register does not change the state of the RX FIFO. Likewise, writing to the TX FIFO push register does not change the state of the TX FIFO. Clearing either of the FIFOs has no effect in the module disable mode. Changes to the DIS_TXF and DIS_RXF fields of the DSPI_MCR has no effect in the module disable mode. In the module disable mode, all status bits and register flags in the DSPI return the correct values when read, but writing to them has no effect. Writing to the DSPI_TCR during module disable mode has no effect. Interrupt and DMA request signals cannot be cleared while in the module disable mode.

30.4.13.3 Slave Interface Signal Gating

The DSPI's module enable signal is used to gate slave interface signals such as address, byte enable, read/write and data. This prevents toggling slave interface signals from consuming power unless the DSPI is accessed. The module enable signal can also be used to gate the clock (ipg_clk_s) to the memory-mapped logic.

30.5 Initialization/Application Information

30.5.1 How to Change Queues

DSPI queues are not part of the DSPI module, but the DSPI includes features in support of queue management. Queues are primarily supported in SPI configuration. This section presents an example of how to change queues for the DSPI.

1. The last command word from a queue is executed. The EOQ bit in the command word is set to indicate to the DSPI that this is the last entry in the queue.
2. At the end of the transfer, corresponding to the command word with EOQ set is sampled, the EOQ flag (EOQF) in the DSPI_SR is set.
3. Setting the EOQF flag disables both serial transmission and reception of data, putting the DSPI in the STOPPED state. The TXRXS bit is negated to indicate the STOPPED state.
4. The eDMA continues to fill TX FIFO until it is full or step 5 occurs.
5. Disable DSPI DMA transfers by disabling the DMA enable request for the DMA channel assigned to TX FIFO and RX FIFO. This is done by clearing the corresponding DMA enable request bits in the eDMA controller.
6. Ensure all received data in RX FIFO has been transferred to memory receive queue by reading the RXCNT in DSPI_SR or by checking RFDF in the DSPI_SR after each read operation of the DSPI_POPR.
7. Modify DMA descriptor of TX and RX channels for new queues.
8. Flush TX FIFO by writing a 1 to the CLR_TXF bit in the DSPI_MCR, Flush RX FIFO by writing a 1 to the CLR_RXF bit in the DSPI_MCR.
9. Clear transfer count either by setting CTCNT bit in the command word of the first entry in the new queue or via CPU writing directly to SPI_TCNT field in the DSPI_TCR.

10. Enable DMA channel by enabling the DMA enable request for the DMA channel assigned to the DSPI TX FIFO, and RX FIFO by setting the corresponding DMA set enable request bit.
11. Enable serial transmission and serial reception of data by clearing the EOQF bit.

30.5.2 Baud Rate Settings

Table 30-35 shows the baud rate that is generated based on the combination of the baud rate prescaler PBR and the baud rate scaler BR in the DSPI_CTAR*n* registers. The values calculated assume a 100 MHz system frequency and the double baud rate DBR bit is clear.

Table 30-35. Baud Rate Values

		Baud Rate Divider Prescaler Values			
		2	3	5	7
Baud Rate Scaler Values	2	25.0M	16.7M	10.0M	7.14M
	4	12.5M	8.33M	5.00M	3.57M
	6	8.33M	5.56M	3.33M	2.38M
	8	6.25M	4.17M	2.50M	1.79M
	16	3.12M	2.08M	1.25M	893k
	32	1.56M	1.04M	625k	446k
	64	781k	521k	312k	223k
	128	391k	260k	156k	112k
	256	195k	130k	78.1k	55.8k
	512	97.7k	65.1k	39.1k	27.9k
	1024	48.8k	32.6k	19.5k	14.0k
	2048	24.4k	16.3k	9.77k	6.98k
	4096	12.2k	8.14k	4.88k	3.49k
	8192	6.10k	4.07k	2.44k	1.74k
	16384	3.05k	2.04k	1.22k	872
	32768	1.53k	1.02k	610	436

30.5.3 Delay Settings

Table 30-36 shows the values for the Delay after Transfer (T_{DT}) and CS to SCK Delay (T_{CSC}) that can be generated based on the prescaler values and the scaler values set in the DSPI_CTAR*n* registers. The values calculated assume a 100 MHz system frequency. This table does not apply for TSB continuous mode.

Table 30-36. Delay Values

		Delay Prescaler Values			
		1	3	5	7
Delay Scaler Values	2	20.0 ns	60.0 ns	100.0 ns	140.0 ns
	4	40.0 ns	120.0 ns	200.0 ns	280.0 ns
	8	80.0 ns	240.0 ns	400.0 ns	560.0 ns
	16	160.0 ns	480.0 ns	800.0 ns	1.1 µs
	32	320.0 ns	960.0 ns	1.6 µs	2.2 µs
	64	640.0 ns	1.9 µs	3.2 µs	4.5 µs
	128	1.3 µs	3.8 µs	6.4 µs	9.0 µs
	256	2.6 µs	7.7 µs	12.8 µs	17.9 µs
	512	5.1 µs	15.4 µs	25.6 µs	35.8 µs
	1024	10.2 µs	30.7 µs	51.2 µs	71.7 µs
	2048	20.5 µs	61.4 µs	102.4 µs	143.4 µs
	4096	41.0 µs	122.9 µs	204.8 µs	286.7 µs
	8192	81.9 µs	245.8 µs	409.6 µs	573.4 µs
	16384	163.8 µs	491.5 µs	819.2 µs	1.1 ms
	32768	327.7 µs	983.0 µs	1.6 ms	2.3 ms
	65536	655.4 µs	2.0 ms	3.3 ms	4.6 ms

30.5.4 Oak Family Compatibility with the DSPI

Table 30-37 shows the translation of commands written to the TX FIFO command halfword with commands written to the Command Ram of the Oak family QSPI. The table illustrates how to configure the DSPI_CTAR*n* registers to match the default cases for the possible combinations of the Oak Family Control Bits in its command RAM. The defaults for the Oak Family are based on a system clock of 40 MHz. All delay variables below generate the same delay, or as close as possible, from the DSPI 100 MHz system clock that an Oak Family part would generate from its 40 MHz system clock. For other system clock frequencies, the customer can recompute the values using Section 30.5.3, Delay Settings.

- For BITSE = 0 → 8 bits per transfer
- For DT = 0 → 0.425 µs delay: For this value, the closest value in the DSPI is 0.480 µs
- For DSCK = 0 → 1/2 SCK period: For this value, the value for the DSPI is 20 ns

Table 30-37. Oak Family QSPI Compatibility with the DSPI

Oak Family Control Bits DSPI Corresponding Control Bits						Corresponding DSPI_CTAR*n* Register Configuration					
BITSE	CTAS[0]	DT	CTAS[1]	DSCK	CTAS[2]	DSPI_CTAR*n*	FMSZ	PDT	DT	PCSSCK	CSSCK
0		0		0		0	1111	10	0011	00	0000
0		0		1		1	1111	10	0011	user	user
0		1		0		2	1111	user[1]	user	00	0000
0		1		1		3	1111	user	user	user	user
1		0		0		4	user	10	0011	00	0000
1		0		1		5	user	10	0011	user	user
1		1		0		6	user	user	user	00	0000
1		1		1		7	user	user	user	user	user

1 Selected by user

30.5.5 Calculation of FIFO Pointer Addresses

The user has complete visibility of the TX and RX FIFO contents through the FIFO registers, and valid entries can be identified through a memory-mapped pointer and a memory-mapped counter for each FIFO. The pointer to the first-in entry in each FIFO is memory mapped. For the TX FIFO the first-in pointer is the transmit next pointer (TXNXTPTR). For the RX FIFO the first-in pointer is the pop next pointer (POPNXTPTR). Figure 30-43 illustrates the concept of first-in and last-in FIFO entries along with the FIFO counter. The TX FIFO is chosen for the illustration, but the concepts carry over to the RX FIFO. See Section 30.4.3.4, Transmit First-In First-Out (TX FIFO) Buffering Mechanism, and Section 30.4.3.5, Receive First-In First-Out (RX FIFO) Buffering Mechanism, for details on the FIFO operation.

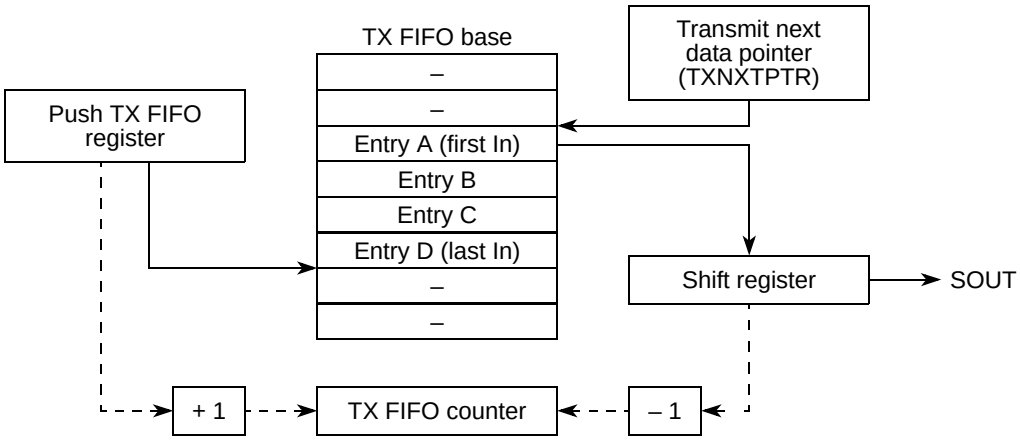


Figure 30-43. TX FIFO Pointers and Counter

30.5.5.1 Address Calculation for the First-in Entry and Last-in Entry in the TX FIFO

The memory address of the first-in entry in the TX FIFO is computed by the following equation:

$$\text{First-in Entry Address} = \text{TX FIFO Base} + (4 \times \text{TXNXTPTR}) \quad \textit{Eqn. 30-10}$$

The memory address of the last-in entry in the TX FIFO is computed by the following equation:

$$\text{Last-in Entry address} = \text{TX FIFO Base} + 4 \times \text{modulo TX FIFO depth}(\text{TXCTR} + \text{TXNXTPTR} - 1) \quad \textit{Eqn. 30-11}$$

where:

TX FIFO base: base address of TX FIFO
TXCTR: TX FIFO counter
TXNXTPTR: transmit next pointer
TX FIFO depth: transmit FIFO depth

30.5.5.2 Address Calculation for the First-in Entry and Last-in Entry in the RX FIFO

The memory address of the first-in entry in the RX FIFO is computed by the following equation:

$$\text{First-in Entry Address} = \text{RX FIFO Base} + (4 \times \text{POPNXTPTR}) \quad \textit{Eqn. 30-12}$$

The memory address of the last-in entry in the RX FIFO is computed by the following equation:

$$\text{Last-in Entry address} = \text{RX FIFO Base} + 4 \times \text{modulo RX FIFO depth}(\text{RXCTR} + \text{POPNXTPTR} - 1) \quad \textit{Eqn. 30-13}$$

where:

RX FIFO base: base address of RX FIFO
RXCTR: RX FIFO counter
POPNXTPTR: pop next pointer
RX FIFO depth: receive FIFO depth

Chapter 31
Enhanced Serial Communication Interface (eSCI)

31.1 Introduction

The eSCI allows asynchronous serial communications with peripheral devices and other CPUs. The eSCI has special features that allow the eSCI to operate as a LIN bus master, complying with the LIN 1.3, 2.0, 2.1, and SAE J2602 specification.

31.1.1 Block Diagram

A simplified block diagram of the eSCI illustrates the functionality and interdependence of major blocks (see Figure 31-1).

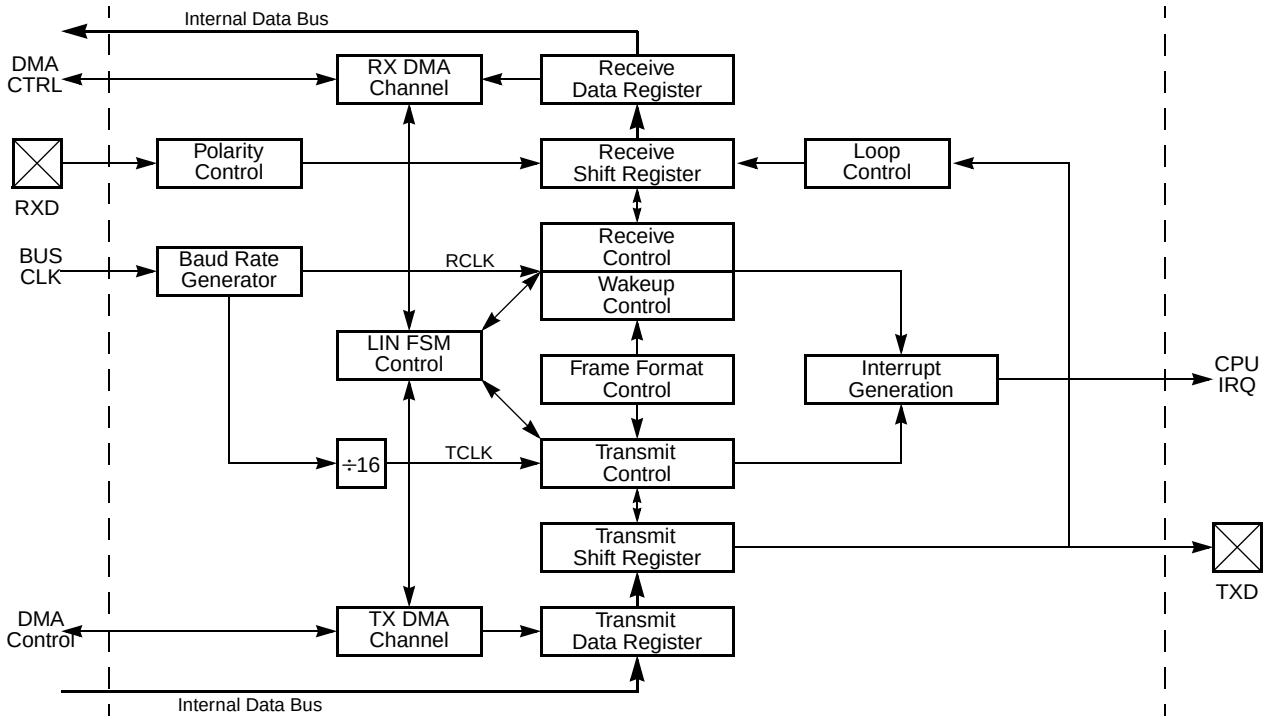


Figure 31-1. eSCI Block Diagram

31.1.2 Features

The eSCI has these major features:

- Full-duplex operation
- Standard mark/space non-return-to-zero (NRZ) format

- 13-bit baud rate selection
- Programmable frame, payload, and character format
- Support of 2 stop bits in receiver path
- Hardware parity generation and checking
  - Programmable even or odd parity
- Programmable polarity of RXD pin
- Separately enabled transmitter and receiver
- Two receiver wake up methods:
  - Idle line wake-up
  - Address mark wake-up
- Interrupt-driven operation with eight flags:
  - Transmitter empty
  - Transmission complete
  - Receiver full
  - Idle receiver input
  - Receiver overrun
  - Noise error
  - Framing error
  - Parity error
- Receiver framing error detection
- 1/16 bit-time noise detection
- 2 channel DMA interface
- LIN support
  - LIN Master Node functionality (master and slave task)
  - Compatible with LIN slaves from revisions 1.x and 2.0 of the LIN standard
  - Detection of Bit Errors, Physical Bus Errors, and Checksum Errors
  - All status bit can generate maskable interrupts
  - Application layer CRC support
  - Programmable CRC polynom
  - Detection and generation of wakeup characters
  - Programmable wakeup delimiter time
  - Programmable slave timeout
  - Can be configured to include header bits in checksum
  - LIN DMA interface

31.1.3 Modes of Operation

The SCI module has two functional operational modes, SCI and LIN mode, and low power modes. The availability of register bits and fields depends on the selected operational mode.

31.1.3.1 SCI Mode

The SCI mode is the default functional operational mode and is described in Section 31.4.5, SCI Mode.

31.1.3.2 LIN Mode

The LIN mode is the second functional operational mode and is described in Section 31.4.6, LIN Mode.

31.1.3.3 Disabled Mode

In the Disabled mode, the eSCI module indicates to the clocking system that all module clocks can be turned off. The eSCI module is in the Disabled mode when the MDIS bit in the eSCI Control Register 2 (eSCI_CR2) is set.

31.1.3.4 Halt Mode

When the eSCI module is in halt mode, it is inactive and indicates to the system that all clocks to the eSCI modules can be turned off.

The system requests the eSCI module to enter the halt mode when the appropriate HLT bit is set in the SIU_HLT0 (for eSCI_A to eSCI_H) or SIU_HLT1 (for eSCI_J to eSCI_M register. When this happens, the eSCI module performs a shut down of the ongoing transmission or reception as described in Section 31.1.3.4.1, Entering Halt Mode from SCI Mode and Section 31.1.3.4.3, Entering Halt Mode from LIN Mode. If the eSCI module is in halt mode and the system stop signal is de-asserted and the LIN bit and the MDIS bit are 0, the eSCI module enters the SCI mode. The TRDE flag is set. No data will be transmitted as long as new data is provided by the application. If the receiver is still enabled, it starts the detection of the start bit.

The system requests the eSCI module to leave the halt mode by de-asserting the HLT bit. When this happens, the eSCI module then enters the operational mode configured by the LIN bit and the MDIS bit. The related sequence is described in Section 31.1.3.4.2, Leaving Halt Mode into SCI Mode and Section 31.1.3.4.4, Leaving Halt Mode into LIN Mode.

31.1.3.4.1 Entering Halt Mode from SCI Mode

If the eSCI module is in SCI mode and the system stop signal is asserted while a SCI frame or character transmission is running, the eSCI module performs a shut down of the transmit process. Therefore, it continues the ongoing frame or character transmission until the last bit of the SCI frame or character has been transmitted. After the end of the transmission, all pending transfer requests are cleared and no more data will be transmitted. None of the transmitter related register flags will be set.

If the eSCI module is in SCI mode and the system stop signal is asserted while a SCI frame or character reception is running, the module stops the reception immediately. None of the receiver related register flags will be set.

If the eSCI module is in SCI mode and the system stop signal is asserted and no transmission or reception is running, the eSCI module enters the halt mode.

31.1.3.4.2 Leaving Halt Mode into SCI Mode

If the eSCI module is in halt mode and the system stop signal is de-asserted and the LIN bit and the MDIS bit are 0, the eSCI module enters the SCI mode. The TRDE flag is set. No data will be transmitted as long as new data is provided by the application. If the receiver is still enabled, it starts the detection of the start bit.

31.1.3.4.3 Entering Halt Mode from LIN Mode

If the eSCI module is in LIN mode and the system stop signal is asserted while a LIN byte field or character transmission is running, the eSCI module performs a shut down of the transmit process. Therefore, it continues the ongoing LIN byte field or character transmission until the last bit of the LIN byte field or character has been transmitted. After the end of the transmission, all pending transfer requests are cleared and no more data will be transmitted. None of the transmitter related register flags will be set.

If the eSCI module is in LIN mode and the system stop signal is asserted while a LIN byte field or character reception is running, the eSCI module aborts the reception immediately. None of the receiver related register flags will be set.

If the eSCI module is in LIN mode and the system stop signal is asserted and no transmission or reception is running, the eSCI module resets the LIN protocol engine to its idle state, resets the write access counter of the LIN Transmit Register (eSCI_LTR) and enters the halt mode.

31.1.3.4.4 Leaving Halt Mode into LIN Mode

If the eSCI module is in halt mode and the system stop signal is de-asserted and the LIN bit is set and the MDIS bit is 0, the eSCI module enters the LIN mode. The TXRDY flag is set. No data will be transmitted as long as new LIN frame header data is provided by the application. If the receiver is still enabled, it starts the detection of the start bit.

31.2 External Signal Description

Each eSCI_*x* module has two external signals: TXD_*x* (transmit data output of eSCI_*x*) and RXD_*x* (receive data input of eSCI_*x*). Refer to Table 3-1 and Section 3.4, Detailed Signal Description, for detailed signal descriptions.

31.3 Memory Map and Registers

This section provides the memory map and a detailed description of the memory mapped registers.

31.3.1 Memory Map

The eSCI memory map is shown in Table 31-1. The address of each register is given as an offset to the eSCI base address. Registers are listed in address order, identified by complete name and mnemonic, and include the type of accesses allowed.

NOTE

eSCI_G, eSCI_H, eSCI_J, eSCI_K, eSCI_L, and eSCI_M are not implemented on the PXN20.

Table 31-1. eSCI Memory Map

Offset from ESCI_BASE eSCI_A = 0xFFFA_0000 eSCI_B = 0xFFFA_4000 eSCI_C = 0xFFFA_8000 eSCI_D = 0xFFFA_C000 eSCI_E = 0xFFFB_0000 eSCI_F = 0xFFFB_4000 eSCI_G = 0xFFFB_8000 eSCI_H = 0xFFFB_C000 eSCI_J = 0xC3FA_0000 eSCI_K = 0xC3FA_4000 eSCI_L = 0xC3FA_8000 eSCI_M = 0xC3FA_C000	Register	Access	Reset Value	Section/Page
0x0000	eSCI_BRR—eSCI baud rate register	R/W	0x0004	31.3.2.1/31-6
0x0002	eSCI_CR1—eSCI control register 1	R/W	0x0000	31.3.2.2/31-6
0x0004	eSCI_CR2—eSCI control register 2	R/W	0xA000	31.3.2.3/31-8
0x0006	eSCI_SDR—eSCI data register	R/W	0x0000	31.3.2.4/31-10
0x0008	eSCI_IFSR1—eSCI interrupt flag and status register 1	R/W	0x8000	31.3.2.5/31-11
0x000A	eSCI_IFSR2—eSCI interrupt flag and status register 2	R/W	0x4000	31.3.2.6/31-12
0x000C	eSCI_LCR1—eSCI LIN control register 1	R/W	0x0000	31.3.2.7/31-13
0x000E	eSCI_LCR2—eSCI LIN control register 2	R/W	0x0000	31.3.2.8/31-15
0x00010	eSCI_LTR— eSCI LIN transmit register	R/W	0x0000	31.3.2.9/31-15
0x0012	Reserved			
0x0014	eSCI_LRR—eSCI LIN receive register	R/W	0x0000	31.3.2.10/31-17
0x0016	Reserved			
0x0018	eSCI_LPR—eSCI LIN CRC polynomial register	R/W	0xC599	31.3.2.11/31-18
0x001A	eSCI_CR3—eSCI control register 3	R/W	0x0000	31.3.2.12/31-18
0x001C–0x3FFF	Reserved			

31.3.2 Register Descriptions

This section consists of register descriptions in address order. Each description includes a standard register diagram with an associated figure number. Writes to a reserved register location do not have any effect and reads of these locations return a 0. Details of register bit and field function follow the register diagrams, in bit order.

31.3.2.1 eSCI Baud Rate Register (eSCI_BRR)

This register provides the control value for the serial baud rate. The baud rate and clock generation is specified in Section 31.4.3, Baud Rate and Clock Generation.

A byte write access to only the upper byte of this register (eSCI_BRR[0:7]) will not change the content of the register. Instead, the written byte is stored internally into a shadow register. A subsequent byte write access to only the lower byte of this register (eSCI_BRR[8:15]) updates the lower byte and copies the contents of the shadow register into the upper byte.

A byte write access to only the lower byte of this register (eSCI_BRR[8:0]) without a preceding byte write access to only the upper byte copies a value of all zeroes into the upper byte.

A word write access to this register updates both the lower and upper byte immediately and is the recommended write access type for this register.

Offset: ESCI_BASE + 0x0000 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	SBR												
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0

Figure 31-2. eSCI Baud Rate Register (eSCI_BRR)

Table 31-2. eSCI_BRR Field Descriptions

Field	Description
SBR	Serial Baud Rate. This field provides the baud rate control value SBR.

31.3.2.2 eSCI Control Register 1 (eSCI_CR1)

This register provides bits to configure the functionality of the module, provides the interrupt enable bits for the interrupt flags provided in the eSCI Interrupt Flag and Status Register 1 (eSCI_IFSR1) and provides the control bits for the transmitter and receiver.

Offset: ESCI_BASE + 0x0002 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	LOOPS	0	RSRC	M	WAKE	ILT	PE	PT	TIE	TCIE	RIE	ILIE	TE	RE	RWU	SBK
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 31-3. eSCI Control Register 1 (eSCI_CR1)

Table 31-3. eSCI_CR1 Field Descriptions

Field	Description
LOOPS	Loop Mode Select. Together with the RSRC control bit, this control bit defines the receiver source mode. The mode coding is defined in Table 31-4 and the modes are described in Section 31.4.5.3.2, Receiver Input Mode Selection.
RSRC	Receiver Source Control. Together with the LOOPS control bit, this control bit defines the receiver source mode. The mode coding is defined in Table 31-4 and the modes are described in Section 31.4.5.3.2, Receiver Input Mode Selection.
M	Frame Format Mode. Together with the M2 bit of the eSCI Control Register 3 (eSCI_CR3), this control bit controls the frame format used. The supported frame formats and the related settings are described in Section 31.4.2, Frame Formats.
WAKE	Receiver Wake-up Condition. This control bit defines the wake-up condition for the receiver. The receiver wake-up is described in Section 31.4.5.5, Multiprocessor Communication. 0 Idle line wake-up. 1 Address mark wake-up
ILT	Idle Line Type. This control bit defines the type of idle line detection for the receiver wake-up. The two types are described in Section 31.4.5.5.1, Idle-Line Wakeup. 0 Idle line detection starts after reception of a low bit. 1 Idle line detection starts after reception of the last stop bit.
PE	Parity Enable. This control bit enables the parity bit generation and checking. The location of the parity bits is shown in Section 31.4.2, Frame Formats. 0 Parity bit generation and checking disabled. 1 Parity bit generation and checking enabled.
PT	Parity Type. This control bit defines whether even or odd parity has to be used. 0 Even parity (even number of ones in character clears the parity bit). 1 Odd parity (odd number of ones in character clears the parity bit).
TIE	Transmitter Interrupt Enable. This bit controls the eSCI_IFSR1[TRDE] interrupt request generation. 0 TDRE interrupt request generation disabled. 1 TDRE interrupt request generation enabled.
TCIE	Transmission Complete Interrupt Enable. This bit controls the eSCI_IFSR1[TC] interrupt request generation. 0 TC interrupt request generation disabled. 1 TC interrupt request generation enabled.
RIE	Receiver Full Interrupt Enable. This bit controls the eSCI_IFSR1[RDRF] interrupt request generation. 0 RDRF interrupt request generation disabled. 1 RDRF interrupt request generation enabled.
ILIE	Idle Line Interrupt Enable. This bit controls eSCI_IFSR1[IDLE] interrupt request generation. 0 IDLE interrupt request generation disabled. 1 IDLE interrupt request generation enabled.
TE	Transmitter Enable. This control bit enables and disables the transmitter. The control features of the transmitter are described in Section 31.4.5.2.1, Transmitter States and Transitions. 0 Transmitter disabled. 1 Transmitter enabled.
RE	Receiver Enable.This control bit enables and disables the receiver. The control features of the receiver are described in Section 31.4.5.3.1, Receiver States and Transitions. 0 Receiver disabled. 1 Receiver enabled.

Table 31-3. eSCI_CR1 Field Descriptions (continued)

Field	Description
RWU	Receiver Wake-Up Mode. This bit controls and indicates the receiver wake-up mode, which is described in Section 31.4.5.5, Multiprocessor Communication. 0 Normal receiver operation. 1 Receiver is in wake-up mode. **Note:** This bit should be set in SCI mode only.
SBK	Send Break Character. This bit controls the transmission of break characters, which is described in Section 31.4.5.2.7, Break Character Transmission. 0 No break characters are transmitted. 1 Break characters are transmitted. **Note:** This bit should be set in SCI mode only.

Table 31-4. Receive Source Mode Selection

LOOPS	RSCR	Receiver Input Mode
0	0	Dual Wire Mode
0	1	Reserved
1	0	Loop Mode
1	1	Single Wire Mode

31.3.2.3 eSCI Control Register 2 (eSCI_CR2)

This register provides bits to configure the functionality of the module, and interrupt enable bits for the interrupt flags provided in eSCI Interrupt Flag and Status Register 1 (eSCI_IFSR1) and control bits for the transmitter and receiver.

Offset: ESCI_BASE + 0x0004 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	MDIS	FBR	BSTP	BERR IE	RX DMA	TX DMA	BRCL	TX DIR	BE SM	BE STP	RX POL	P MSK	ORIE	NFIE	FEIE	PFIE
Reset	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 31-4. eSCI Control Register 2 (eSCI_CR2)

Table 31-5. eSCI_CR2 Field Descriptions

Field	Description
MDIS	Module Disabled Mode. This bit controls the module mode of operation, which is described in Section 31.1.3, Modes of Operation. 0 Module is not in disabled mode. 1 Module is in disabled mode.
FBR	Fast Bit Error Detection. This bit controls the Bit Error Detection mode. 0 Standard bit error detection performed as described in **Note:** 1Fast bit error detection performed as described in This bit is used in LIN mode only.
BSTP	DMA Stop on Bit Error or Physical Bus Error. This bit controls the transmit DMA requests generation in case of bit errors or physical bus errors. Bit errors are indicated by the BERR flag in the Interrupt Flag and Status Register 1 (eSCI_IFSR1) and physical bus errors are indicated by the PBERR flag in the Interrupt Flag and Status Register 2 (eSCI_IFSR2). 0 Transmit DMA requests generated regardless of bit errors or physical bus errors. 1 Transmit DMA requests are *not* generated if eSCI_IFSR1[BERR] or eSCI_IFSR2[PBERR] flags are set. **Note:** This bit is used in LIN mode only.
BERRIE	Bit Error Interrupt Enable. This bit controls the BERR interrupt request generation. 0 BERR interrupt request generation disabled. 1 BERR interrupt request generation enabled.
RXDMA	Receive DMA Control. This bit enables the receive DMA feature. 0 Receive DMA disabled. 1 Receive DMA enabled.
TXDMA	Transmit DMA Control. This bit enables the transmit DMA feature. 0 Transmit DMA disabled. 1 Transmit DMA enabled.
BRCL	Break Character Length. This bit is used to define the length of the break character to be transmitted. The settings are specified in Section 31.4.2.2, Break Character Formats.
TXDIR	TXD pin output enable. This bit determines whether the TXD pin is used as an output. 0 TXD pin is not used as output. 1 TXD pin is used as output. **Note:** This bit is used in Single Wire Mode only.
BESM	Fast Bit Error Detection Sample Mode. This bit defines the sample point for the fast bit error detection mode. 0 Sample point is RS9. 1 Sample point is RS13. **Note:** This bit is used in LIN mode only.
BESTP	Transmit Stop on Bit Error. If this control bit is set, the eSCI stops driving the LIN bus immediately when a Bit Error has been detected, i.e. eSCI_IFSR1[BERR] = 1. Additionally, the eSCI will not start a new byte transmission as long the BERR interrupt flag is set. 0 Transmission is not stopped on bit error. 1 Transmission is stopped on bit error. **Note:** This bit is used in LIN mode only.
RXPOL	RXD Pin polarity. This bit controls the polarity of the RXD pin. See Section 31.4.2.1.1, Inverted Data Frame Formats. 0 Normal Polarity. 1 Inverted Polarity.

Table 31-5. eSCI_CR2 Field Descriptions (continued)

Field	Description
PMSK	Parity Bit Masking. This bit defines whether the received parity bit is presented in the related bit position in the SCI Data Register (eSCI_SDR). 0 The received parity bit is presented in the bit position related to the parity bit. 1 The value 0 is presented in the bit position related to the parity bit.
ORIE	Overrun Interrupt Enable. This bit controls the eSCI_IFSR1[OR] interrupt request generation. 0 OR interrupt request generation disabled. 1 OR interrupt request generation enabled.
NFIE	Noise Interrupt Enable. This bit controls the eSCI_IFSR1[NF] interrupt request generation. 0 NF interrupt request generation disabled. 1 NF interrupt request generation enabled.
FEIE	Frame Error Interrupt Enable. This bit controls the eSCI_IFSR1[FE] interrupt request generation. 0 FE interrupt request generation disabled. 1 FE interrupt request generation enabled.
PFIE	Parity Error Interrupt Enable. This bit controls the eSCI_IFSR1[PF] interrupt request generation. 0 PF interrupt request generation disabled. 1 PF interrupt request generation enabled.

31.3.2.4 eSCI Data Register (eSCI_DR)

This register is used to provide transmit data and retrieve received data in SCI mode. In LIN mode any write access to this register is ignored and any read access returns all 0. In case of data transmission this register is used to provide a part of the transmit data. In case of data reception this register provides a part of the received data and related error information.

If the application writes to the lower byte of this register (eSCI_SDR[7:0]), the internal commit flag iCMT, which is not visible to the application, is set to indicate that the register has been updated and ready to transmit new data.

If the application reads from the lower byte of this register (eSCI_SDR[7:0]), a signal is send to the internal receiver unit to indicate that the register was r

Offset: ESCI_BASE + 0x0006 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	RN	TN	ERR	0	RD[11:8]				RD[7]	RD[6:0]						
W									TD[7]	TD[6:0]						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 31-5. eSCI Data Register (eSCI_DR)

Table 31-6. eSCI_DR Field Descriptions

Field	Description
RN	Received Most Significant Bit. The semantic of this bit depends on the frame format selected by eSCI_CR3[M2], eSCI_CR1[M], and eSCI_CR1[PE]. [M2 = 0,M = 1,PE = 0]: value of received data bit 8 or address bit. [M2 = 0,M = 1,PE = 1]: value of received parity bit if eSCI_CR2[PMSK] = 0, 0 otherwise. [M2 = 1,M = 0,PE = 1]: value of received parity bit if eSCI_CR2[PMSK] = 0, 0 otherwise. [M2 = 1,M = 1,PE = 1]: value of received parity bit if eSCI_CR2[PMSK] = 0, 0 otherwise. It is 0 for all other frame formats.
TN	Transmit Most Significant Bit. The semantic of this bit depends on the frame format selected by eSCI_CR3[M2], eSCI_CR1[M], and eSCI_CR1[PE]. [M2 = 0,M = 1,PE = 0]: value to be transmitted as data bit 8 or address bit. It is not used for all other frame formats.
ERR	Receive Error Bit. This bit indicates the occurrence of the errors selected by the Control Register 3 (eSCI_CR3) during the reception of the frame presented in SCI Data Register (eSCI_SDR). In case of an overrun error for subsequent frames this bit is set too. 0 None of the selected errors occured. 1 At least one of the selected errors occured.
RD[11:8]	Received Data. The semantic of this field depends on the frame format selected by eSCI_CR3[M2] and eSCI_CR1[M]. [M2 = 1,M = 1]: value of the received data bits 11:8. (R*n* = BIT*n*). It is all 0 for all other frame formats.
RD7	Received Bit 7. The semantic of this bit depends on the format selected by eSCI_CR3[M2], eSCI_CR1[M], and eSCI_CR1[PE]. [M2 = 0,M = 0,PE = 0]: value of received bit 7 or ADDR bit. [M2 = 0,M = 0,PE = 1]: value of received parity bit if eSCI_CR2[PMSK] = 0, 0 otherwise. For all other frame formats it is the value of received bit 7.
TD7	Transmit Bit 7. The semantic of this bit depends on the format selected by eSCI_CR3[M2], eSCI_CR1[M], and eSCI_CR1[PE]. [M2 = 0,M = 0,PE = 0]: value of transmit bit 7 or ADDR bit. [M2 = 0,M = 0,PE = 1]: not used. Parity bit is generated internally before transmission. For all other frame formats it is the value of transmit bit 7.
RD[6:0]	Received bits 6 to 0. Value of received BIT*n* is shown in bit RD*n*
TD[6:0]	Transmit bits 6 to 0. Value of bit TD*n* is transmitted in BIT*n*.

31.3.2.5 eSCI Interrupt Flag and Status Register 1 (eSCI_IFSR1)

This register provides interrupt flags that indicate the occurrence of module events. The related interrupt enable bits are located in Control Register 1 (eSCI_CR1) and Control Register 2 (eSCI_CR2).

Offset: ESCI_BASE + 0x0008 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	TDRE	TC	RDRF	IDLE	OR	NF	FE	PF	0	0	0	BERR	0	0	TACT	RACT
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c				w1c				
Reset	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 31-6. eSCI Interrupt Flag and Status Register 1 (eSCI_IFSR1)

Table 31-7. eSCI_IFSR1 Field Descriptions

Field	Description
TDRE	Transmit Data Register Empty Interrupt Flag. This interrupt flag is set when the content of the SCI Data Register (eSCI_SDR) is transferred into the internal shift register. **Note:** This flag is set in SCI mode only.
TC	Transmit Complete Interrupt Flag. This interrupt flag is set when a frame, break or idle character transmission has been completed and no data were written into SCI Data Register (eSCI_SDR) after the last setting of the TDRE flag and the SBK bit in Control Register 1 (eSCI_CR1) is 0. This flag is set in LIN mode if the preamble was transmitted after enabling the transmitter.
RDRF	Receive Data Register Full Interrupt Flag. This interrupt flag is set when the payload data of a received frame is transferred into the SCI Data Register (eSCI_SDR). **Note:** This flag is set in SCI mode only.
IDLE	Idle Line Flag. This flag is set when an idle character was detected and the receiver is not in the wakeup state. **Note:** This flag is set in SCI mode only.
OR	Overrun Flag. This flag is set when an overrun was detected as described in Section 31.4.5.3.11, Receiver Overrun. **Note:** This flag is set in SCI mode only.
NF	Noise Interrupt Flag. This flag is set when the payload data of a received frame was transferred into the SCI Data Register (eSCI_SDR) or LIN Receive Register (eSCI_LRR) and the receiver has detected noise during the reception of that frame, as described in Section 31.4.5.3.13, Bit Sampling.
FE	Framing Error Interrupt Flag. This interrupt flag is set when the payload data of a received frame was transferred into the SCI Data Register (eSCI_SDR) or LIN Receive Register (eSCI_LRR) and the receiver has detected a framing error during the reception of that frame, as described in Section 31.4.5.3.18, Stop Bit Verification.
PF	Parity Error Interrupt Flag. This interrupt flag is set when the payload data of a received frame was transferred into the SCI Data Register (eSCI_SDR) and the receiver has detected a parity error for the character, as described in Section 31.4.5.4, Reception Error Reporting. **Note:** This flag is set in SCI mode only.
BERR	Bit Error Flag. This flag is set when a bit error was detected as described in Section 31.4.6.5.3, Standard Bit Error Detection. **Note:** This flag is set in LIN mode only.
TACT	Transmitter Active. The status bit is set as long as the transmission of a frame or special character is ongoing. 0 No transmission in progress. 1 Transmission in progress.
RACT	Receiver Active. The bit is set 3 receiver clock (RCLK) cycles after the successful qualification of a start bit. This bit is cleared when an idle character is detected. 0 No reception in progress. 1 Reception in progress.

31.3.2.6 eSCI Interrupt Flag and Status Register 2 (eSCI_IFSR2)

This register provides interrupt flags that indicate the occurrence of LIN related events. The related interrupt enable bits are located in eSCI LIN Control Register 1 (eSCI_LCR1) and eSCI LIN Control Register 2 (eSCI_LCR2). All interrupt flags in this register are set in LIN mode only.

Offset: ESCI_BASE + 0x000A Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	RX RDY	TX RDY	L WAKE	STO	PB ERR	CERR	CK ERR	FRC	0	0	0	0	0	0	UREQ	OVFL
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c							w1c	w1c
Reset	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 31-7. eSCI Interrupt Flag and Status Register 2 (eSCI_IFSR2)

Table 31-8. LINSTAT1 Field Descriptions

Field	Description
RXRDY	Receive Data Ready Interrupt Flag. This interrupt flag is set when the payload data of a received frame is transferred into the LIN Receive Register (eSCI_LRR).
TXRDY	Transmit Data Ready Interrupt Flag. This interrupt flag is set when the content of the LIN Transmit Register (eSCI_LTR) is processed by the LIN PE either to generate a frame header or to transmit frame data.
LWAKE	LIN Wakeup Received Flag. This interrupt flag is set when a LIN wakeup character is received, as described in Section 31.4.6.6, LIN Wakeup.
STO	Slave Timeout Interrupt Flag. This interrupt flag is set when a Slave-Not-Responding-Error is detected. A detailed description is given in Section 31.4.6.5.5, Slave-Not-Responding-Error Detection.
PBERR	Physical Bus Error Flag. This interrupt flag is set when the receiver input remains unchanged for at least 31 RCLK clock cycles after the start of a byte transmission, as described in Section 31.4.6.5, LIN Error Reporting.
CERR	CRC Error Flag. This interrupt flag is set when an incorrect CRC pattern was detected for a received LIN frame.
CKERR	Checksum Error Flag. This interrupt flag is set when a checksum error was detected for a received LIN frame.
FRC	Frame Complete Flag. This interrupt flag is set when a LIN frame was completely transmitted or received.
UREQ	Unrequested Data Received Flag. This interrupt flag is set when unrequested activity has been detected on the LIN bus, as described in Section 31.4.6.5, LIN Error Reporting.
OVFL	Overflow Flag. This interrupt flag is set when an overflow as described in Section 31.4.6.5.8, Overflow Detection, was detected.

31.3.2.7 eSCI LIN Control Register 1 (eSCI_LCR1)

This register provides control bits to control and configure the LIN hardware. This register provides the interrupt enable bits for the interrupt flags in Interrupt Flag and Status Register 2 (eSCI_IFSR2).

Offset: ESCI_BASE + 0x000C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	LRES	WU	WUD		0	0	PRTY	LIN	RXIE	TXIE	WUIE	STIE	PBIE	CIE	CKIE	FCIE
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 31-8. eSCI LIN Control Register 1 (eSCI_LCR1)

Table 31-9. eSCI_LCR1 Field Descriptions

Field	Description
LRES	LIN FSM Resync. This bit controls the state of the LIN protocol engine. 0 LIN protocol engine in normal mode. 1 LIN protocol engine hold in initial state.
WU	LIN Bus Wake-Up Trigger. This bit is used to trigger the generation of a wake-up signal on the LIN bus, as described in Section 31.4.6.6, LIN Wakeup. 0 Write has no effect. 1 Write triggers the generation of a wakeup signal.
WUD	LIN Bus Wake-Up Delimiter Time. This field determines how long the LIN protocol engine waits after the end of the transmitted wakeup signal, before starting the next LIN frame transmission. 00 4 bit times. 01 8 bit times. 10 32 bit times. 11 64 bit times.
PRTY	Parity Generation Control. This bit controls the generation of the two parity bits in the LIN header. 0 Parity bits generation disabled. 1 Parity bits generation enabled.
LIN	LIN Mode Control. This bit controls whether the device is in SCI or LIN Mode. 0 SCI Mode. 1 LIN Mode.
RXIE	Receive Data Ready Interrupt Enable. This bit controls the eSCI_IFSR2[RXRDY] interrupt request generation. 0 RXRDY interrupt request generation disabled. 1 RXRDY interrupt request generation enabled.
TXIE	Transmit Data Ready Interrupt Enable. This bit controls the eSCI_IFSR2[TXRDY] interrupt request generation. 0 TXRDY interrupt request generation disabled. 1 TXRDY interrupt request generation enabled.
WUIE	LIN Wakeup Received Interrupt Enable. This bit controls the eSCI_IFSR2[LWAKE] interrupt request generation. 0 LWAKE interrupt request generation disabled. 1 LWAKE interrupt request generation enabled.
STIE	Slave Timeout Flag Interrupt Enable. This bit controls the eSCI_IFSR2[STO] interrupt request generation. 0 STO interrupt request generation disabled. 1 STO interrupt request generation enabled.
PBIE	Physical Bus Error Interrupt Enable. This bit controls the eSCI_IFSR2[PBERR] interrupt request generation. 0 PBERR interrupt request generation disabled. 1 PBERR interrupt request generation enabled.
CIE	CRC Error Interrupt Enable. This bit controls the eSCI_IFSR2[CERR] interrupt request generation. 0 CERR interrupt request generation disabled. 1 CERR interrupt request generation enabled.
CKIE	Checksum Error Interrupt Enable. This bit controls the eSCI_IFSR2[CKERR] interrupt request generation. 0 CKERR interrupt request generation disabled. 1 CKERR interrupt request generation enabled.
FCIE	Frame Complete Interrupt Enable. This bit controls the eSCI_IFSR2[FRC] interrupt request generation. 0 FRC interrupt request generation disabled. 1 FRC interrupt request generation enabled.

31.3.2.8 eSCI LIN Control Register 2 (eSCI_LCR2)

This register provides the interrupt enable bits for the interrupt flags in Interrupt Flag and Status Register 2 (eSCI_IFSR2).

Offset: ESCI_BASE + 0x000E Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	UQIE	OFIE	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 31-9. eSCI LIN Control Register 2 (eSCI_LCR2)

Table 31-10. eSCI_LCR2 Field Descriptions

Field	Description
UQIE	Unrequested Data Received Interrupt Enable. This bit controls the eSCI_IFSR2[UREQ] interrupt request generation. 0 UREQ interrupt request generation disabled. 1 UREQ interrupt request generation enabled.
OFIE	Overflow Interrupt Enable. This bit controls the LINSTAT2[OVFL] interrupt request generation. 0 OVFL interrupt request generation disabled. 1 OVFL interrupt request generation enabled.

31.3.2.9 eSCI LIN Transmit Register (eSCI_LTR)

This register is used by the application to initiate the LIN frame header generation for both LIN TX frames and LIN RX frames. If a LIN TX frame is generated, this register is used to provide the payload data for the LIN TX frame.

If the application initiates a LIN TX frame transfer, i.e the TD bit is set to 1, the content and usage shown in LIN Transmit Register (eSCI_LTR) — LIN TX Frame Generation applies (Figure 31-10). The initiation and transmit of a TX frame is described in Section 31.4.6.3, LIN TX Frame Generation.

If the application initiates an LIN RX frame, i.e the TD bit is set to 0, the content and usage shown in LIN Transmit Register (eSCI_LTR) — LIN RX Frame Generation applies (Figure 31-11). The initiation and transmit of a RX frame is described in Section 31.4.6.4, LIN RX Frame Generation.

Each write access to this register increments the internal write access counter and enables the writing to the next field. The write access counter is reset if:

- The LIN PE is in the idle state (eSCI_LCR1[LRES] = 1)
- A LIN TX frame was completely transmitted (eSCI_IFSR1[FRC] was set to 1)
- A LIN RX frame was completely received (eSCI_IFSR1[FRC] was set to 1)
- The module has entered halt mode.

Offset: ESCI_BASE + 0x0010 Access: User write-only (TD = 1)

Byte	0	1	2	3	4	5	6	7
R								
1st W	P[1:0]		ID[5:0]					
2nd W	LEN							
3rd W	CSM	CSE	CRC	TD (= 1)	TO (ignored)			
4th+ W	DATA							
Reset	0	0	0	0	0	0	0	0

Figure 31-10. eSCI LIN TX Register (eSCI_LTR) — TX Frame Generation

Offset: ESCI_BASE + 0x0010 Access: User write-only (TD = 0)

Byte	0	1	2	3	4	5	6	7
R								
1st W	P[1:0]		ID[5:0]					
2nd W	LEN							
3rd W	CSM	CSE	CRC	TD (= 0)	TO[11:8]			
4th W	TO[7:0]							
Reset	0	0	0	0	0	0	0	0

Figure 31-11. eSCI LIN TX Register (eSCI_LTR) — RX Frame Generation

The content and usage of the eSCI LIN TX Register (eSCI_LTR) depends on the transfer direction of initiated frame. If the application initiates a TX frame transfer, i.e., the TD bit is set to 1, the content and usage shown in eSCI LIN TX Register (eSCI_LTR) — TX Frame Generation register (Figure 31-10) applies. If the application initiates an RX frame, i.e., the TD bit is set to 0, the content and usage shown in eSCI LIN TX Register (eSCI_LTR) — RX Frame Generation register (Figure 31-11) applies.

The initiation and transmit of a TX frame is described in Section 31.4.6.3, LIN TX Frame Generation. The initiation and transmit of a RX frame is described in Section 31.4.6.4, LIN RX Frame Generation.

NOTE

When the eSCI module is in LIN mode and transmits or receives a LIN frame, if the CPU requests Stop Mode, and the Stop Mode is left, a subsequent triggered LIN RX Frame reception may hang. The module will never assert the eSCI_IFSR2[RXRDY] and eSCI_IFSR2[TXRDY] flags. The application should ensure that no LIN transmission is running before it requests Stop Mode by checking the status of the eSCI_IFSR1[TACT] and eSCI_IFSR1[RACT] status flags.

Table 31-11. eSCI_LTR Field Descriptions

Field	Description
P[1:0]	Identifier Parity. This field provides the identifier parity that is used to create the protected identifier if the automatic identifier parity generation is disabled, i.e., the PRTY bit in the eSCI LIN Control Register 1 (eSCI_LCR1) is 0.
ID[5:0]	Identifier. This field is used for the identifier field in the protected identifier.
LEN	Frame Length. This field defines the number of data bytes to be transmitted or received.
CSM	Checksum Model. This bit controls the checksum calculation model used. 0 Classic Checksum Model (LIN 1.3). 1 Enhanced Checksum Model (LIN 2.0).
CSE	Checksum Enable. This bit control the generation and checking of the checksum byte. 0 No generation and checking of checksum byte. 1 Generation and checking of checksum byte.
CRC	CRC Enable. This bit controls the generation of checking standard or enhanced LIN frames, which are described in Section 31.4.6.2, LIN Frame Formats. 0 Standard LIN frame generation and checking. 1 Enhanced LIN frame generation and checking.
TD	Transfer Direction. This bit control the transfer direction of the data, CRC, and checksum byte fields. 0 Data, CRC, and Checksum byte fields received, described in Section 31.4.6.4, LIN RX Frame Generation. 1 Data, CRC, and Checksum byte fields transmitted, described in Section 31.4.6.3, LIN TX Frame Generation.
TO	Timeout Value. The content of the field depends on the transfer direction. RX frame: Defines the time available for a complete RX frame transfer, as described in Section 31.4.6.5.5, Slave-Not-Responding-Error Detection. TX frame: Must be set to 0.
DATA	Transmit Data. Data bits for transmission.

31.3.2.10 eSCI LIN Receive Register (eSCI_LRR)

This register provides the data bytes of received LIN RX frames.

Offset: ESCI_BASE + 0x0014 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	D								0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 31-12. eSCI LIN Receive X Register (eSCI_LRR)

Table 31-12. LINRX Field Descriptions

Field	Description
D	Receive Data. This field provides the data bytes of received LIN RX frames.

31.3.2.11 eSCI LIN CRC Polynomial Register (eSCI_LPR)

This register provides the CRC polynom for generation and processing of CRC-enhanced LIN frames.

Offset: ESCI_BASE + 0x0018 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	P															
Reset	1	1		0	0	1	0	1	1	0	0	1	1	0	0	1

Figure 31-13. eSCI LIN CRC Polynomial Register (eSCI_LPR)

Table 31-13. eSCI_LPR Field Descriptions

Field	Description
P*n*	Polynomial bit x^{Pn}. Used to define the LIN polynomial. The standard is $x^{15} + x^{14} + x^{10} + x^8 + x^7 + x^4 + x^3 + 1$ (the polynomial used for the CAN protocol).

31.3.2.12 eSCI Control Register 3 (eSCI_CR3)

This register is used to control the frame formats and the generation of the ERR bit in the SCI Data Register (eSCI_SDR).

Offset: ESCI_BASE + 0x001A Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	SYNM	EROE	ERFE	ERPE	M2	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 31-14. eSCI Control Register 1 (eSCI_CR1)

Table 31-14. eSCI_CR1 Field Descriptions

Field	Description
SYNM	Synchronization Mode. This bit controls the synchronization mode of the receiver. The synchronization modes are described in Section 31.4.5.3.14, Bit Synchronization. 0 Synchronization performed at each falling data bit edge. 1 Synchronization performed at start bit qualification only.
EROE	ERR flag overrun enable. 0 SCIDRH[ERR] flag not affected by overrun detection. 1 SCIDRH[ERR] flag is set on overrun detection during frame reception.
ERFE	ERR flag frame error enable. 0 eSCI_SDR[ERR] flag not affected by frame error detection. 1 eSCI_SDR[ERR] flag is set on frame error detection for the data provided in eSCI_SDR.

Table 31-14. eSCI_CR1 Field Descriptions (continued)

Field	Description
ERPE	ERR flag parity error enable. 0 eSCI_SDR[ERR] flag not affected by parity error detection. 1 eSCI_SDR[ERR] flag is set on parity error detection for the data provided in eSCI_SDR.
M2	Frame Format Mode 2. Together with the M bit of the Control Register 1 (eSCI_CR1), this bit controls the frame format used. The supported frame formats and the related settings are defined in Section 31.4.2, Frame Formats.

31.4 Functional Description

This section provides a complete functional description of the eSCI block, detailing the operation of the design from the end user perspective in a number of subsections.

31.4.1 Module Control

The operational mode of the module is controlled by the MDIS bit in the eSCI Control Register 2 (eSCI_CR2). The module can transmit and receive data when it is enabled, i.e., MDIS = 1.

31.4.2 Frame Formats

The eSCI module uses the standard NRZ mark/space data format. The eSCI supports three basic frame types, which are the data frames, break characters, and idle characters.

31.4.2.1 Data Frame Formats

Each data frame contains a character that is surrounded by a start bit, an optional parity or address bit, and one or two stop bits. The supported data frame formats for transmission and reception are specified in Table 31-15. The supported data frame formats for reception only are specified in Table 31-16.

Table 31-15. Supported Data Frame Formats for RX and TX

Control				Frame Content				
eSCI_CR3	eSCI_CR1				Payload Bits			
M2	M	PE	WAKE	Start Bits	Character Bits	Address Bits[1]	Parity Bits	Stop Bits
LIN byte fields (Figure 31-15)								
0	0	0	0	1	8	0	0	1
SCI frames (8 payload bits)(Figure 31-16)								
0	0	0	0	1	8	0	0	1
0	0	0	1	1	7	1	0	1
0	0	1	0	1	7	0	1	1
SCI frames (9 payload bits) (Figure 31-17)								
0	1	0	0	1	9	0	0	1
0	1	0	1	1	8	1	0	1
0	1	1	0	1	8	0	1	1

1 The address bit identifies the frame as an address character. See Section 31.4.5.5, Multiprocessor Communication.

Table 31-16. Supported Data Frame Formats for RX only

Control				Frame Content				
eSCI_CR3	eSCI_CR1				Payload Bits			
M2	M	PE	WAKE	Start Bits	Character Bits	Address Bits	Parity Bits	Stop Bits
SCI frames (2 stop bits) (see Figure 31-18)								
1	0	1	0	1	8	0	1	2
1	1	1	0	1	12	0	1	2

The structure of the LIN frames in normal polarity is shown in Figure 31-15.

Figure 31-15. LIN Byte Field Format

The structures of the supported SCI frame formats with 8 payload bits are shown in Figure 31-16.

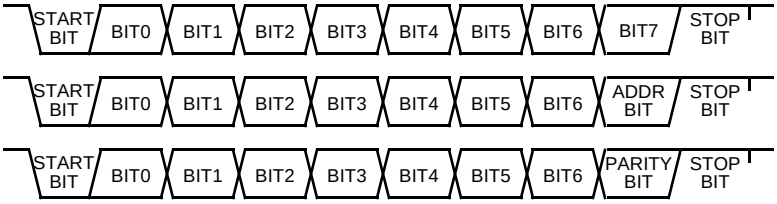


Figure 31-16. SCI Frame Formats (8 Payload Bits)

The structures of the supported SCI frame formats with 9 payload bits are shown in Figure 31-17.

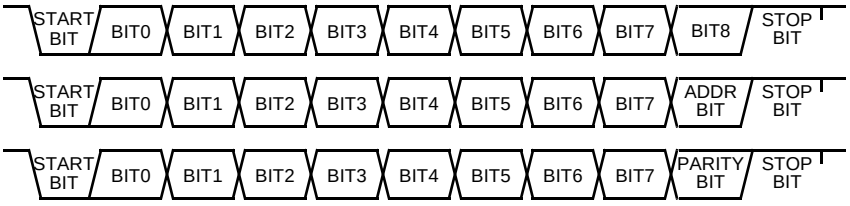


Figure 31-17. SCI Frame Formats (9 Payload Bits)

The structures of the supported SCI frame formats with 2 stop bits in normal polarity are shown in Figure 31-18. This frame format is supported for reception only.

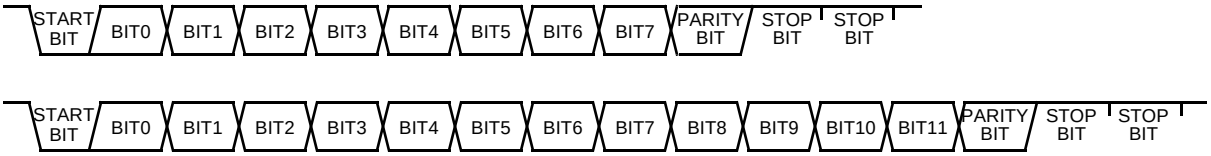


Figure 31-18. SCI Frame Formats (2 Stop Bits)

31.4.2.1.1 Inverted Data Frame Formats

The structures of the supported data frame formats in inverted polarity are shown in Figure 31-19. These frame types are supported for reception only. The polarity of the RXD pin is controlled by the RXPOL bit in the eSCI_ Control Register 2 (eSCI_CR2).

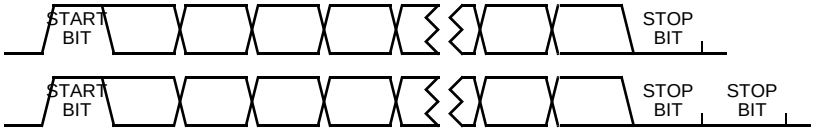


Figure 31-19. Inverted SCI Frame Formats

31.4.2.2 Break Character Formats

The supported break character formats are specified in Table 31-17.

Table 31-17. Supported Break Character Formats

Control[1]			Break Character Content		
eSCI_CR3	eSCI_CR1	eSCI_CR2	Start Bit	Character Bits	Delimit Bits
M2	M	BRCL			
LIN Break Symbol (see Figure 31-20)					
0	0	0	1	9	1
0	0	1	1	12	1
SCI Break Character (see Figure 31-21)					
0	0	0	1	9	0
0	0	1	1	12	0
0	1	0	1	10	0
0	1	1	1	13	0

1 All codings not listed are reserved and must not be used.

The structure and content of the LIN break symbols is shown in Figure 31-20.

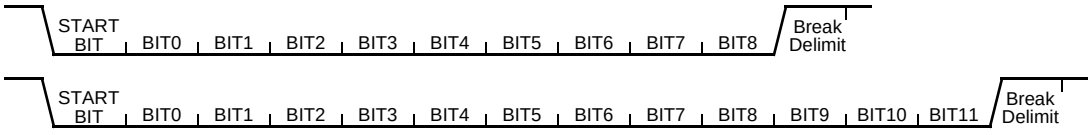


Figure 31-20. LIN Break Symbol Format

The structure and content of the SCI break characters is shown in Figure 31-21.

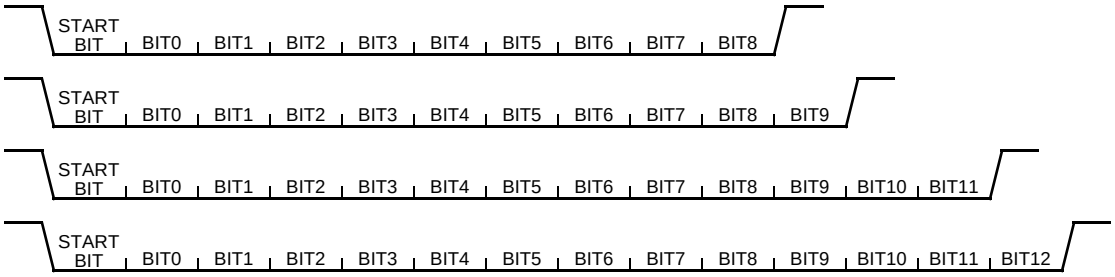


Figure 31-21. SCI Break Character Formats

31.4.2.3 Idle Character Formats

An idle character is a sequence of bits with the value 1. The supported idle character formats are specified in Table 31-18. The preamble has the same structure and content as an idle character.

Table 31-18. Supported Idle Character Formats

Control		Idle Character Length
eSCI_CR3[M2]	eSCI_CR1[M]	
SCI Idle Characters (see Figure 31-22)		
0	0	10
0	1	11
1	0	12
1	1	16

The structure and content of the SCI idle characters is shown in Figure 31-22.

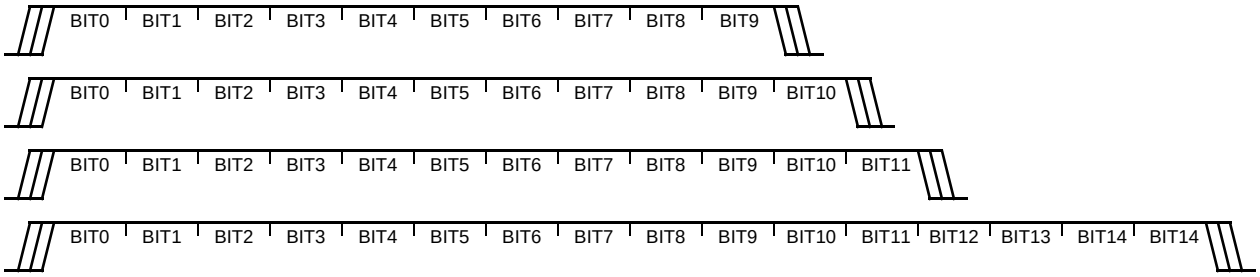


Figure 31-22. SCI Idle Character Formats

31.4.3 Baud Rate and Clock Generation

A 13-bit modulus counter in the baud rate generator derives the baud rate for both the receiver and the transmitter. The value written to the SBR field in the eSCI Baud Rate Register (eSCI_BRR) determines the module clock divisor. The baud rate clock is synchronized with the bus clock and drives the receiver. The baud rate clock divided by 16 drives the transmitter. The receiver has an acquisition rate of 16 samples per bit time.

The baud rate generator is enabled when the TE bit or RE bit in the eSCI Control Register 1 (eSCI_CR1) is set to 1 for the first time. The baud rate generator is disabled when SBR = 0.

Baud rate generation is subject to one source of error:

- Integer division of the module clock may not give the exact required target baud rate.

Figure 31-19 lists some examples of achieving target baud rates with a module clock frequency of MCLK = 10.2 MHz.

Table 31-19. Baud Rates Error Example (MCLK = 10.2 MHz)

eSCI_BRR[SBR]	RCLK (Hz)	TCLK (Hz)	Target Baud Rate	Error (%)
17	600,000.0	37,500.0	38,400	2.3
33	309,090.9	19,318.2	19,200	.62
66	154,545.5	9659.1	9600	.62
133	76,691.7	4793.2	4800	.14
266	38,345.9	2396.6	2400	.14
531	19,209.0	1200.6	1200	.11
1062	9604.5	600.3	600	.05
2125	4800.0	300.0	300	.00
4250	2400.0	150.0	150	.00
5795	1760.1	110.0	110	.00

31.4.3.1 Module Clock

The module clock MCLK is derived from the system bus clock. It has the same phase and frequency.

31.4.3.2 Transmitter Clock

The transmitter clock TCLK is used to drive the data to the serial bus via the TXD pin. It is derived from the system bus clock by the baud rate generator. The baud rate generator is controlled by the value of the SBR[12:0] field in the eSCI Baud Rate Register (eSCI_BRR). The frequency of the transmitter clock is determined by Equation 31-1 and defines the length of the transmitted bits, which is denoted as the *bit time*.

$$f_{TCLK} = \frac{f_{MCLK}}{16 \cdot SBR[12:0]} \qquad \textit{Eqn. 31-1}$$

31.4.3.3 Receiver Clock

The receiver clock RCLK is used to sample the data received on the RXD or TXD pin. It is derived from the system bus clock by the baud rate generator. The baud rate generator is controlled by the value of the SBR[12:0] field in the eSCI Baud Rate Register (eSCI_BRR). The frequency of the receiver sample clock is determined by Equation 31-2.

$$f_{RT} = \frac{f_{MCLK}}{SBR[12:0]} \qquad \textit{Eqn. 31-2}$$

Since the frequency of the receiver clock is 16 times the frequency of the transmitter clock, each bit is sampled 16 times. Each of the 16 samples of a bit has a sample number assigned, which is defined by the receiver sample counter RSC. The *n*th sample is denoted by RS*n*. The receiver sample counter RSC is updated with each rising edge of the receiver clock RCLK.

31.4.4 Baud Rate Tolerance

A transmitting device may be operating at a baud rate below or above the receiver baud rate. Accumulated bit time misalignment can cause one of the three stop bit data samples RS8, RS9, and RS10 to fall outside the actual stop bit. A noise error occurs if the stop bit sample RS8, RS9, and RS10 samples are not all the same logical value 1. A framing error occurs if the receiver clock is misaligned in such a way that the majority of the RS8, RS9, and RS10 stop bit samples are a logic zero.

31.4.4.1 Faster Receiver Tolerance

In this case the receiver has a higher baud rate than the transmitter, thus the stop bit sampling starts already in the last transmitted payload bit. To ensure the correct noise- and framing error-free reception of the stop bit, the samples RS8, RS9, and RS10 must be located in the transmitted stop bit as shown in Figure 31-23.

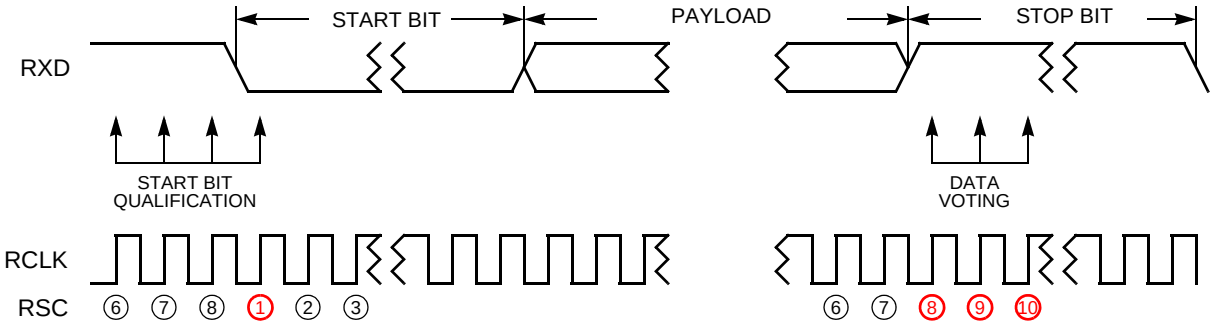


Figure 31-23. Faster Receiver

The maximum tolerance that ensures error-free reception can be calculated with the assumption, that RS7 is sampled during the last transmitted payload bit and RS8 is sampled in the stop bit.

For an frame with *n* payload bits the transmitter starts the transmission of the stop bit

$$tx_{STOP} = (n + 1) \cdot 16 \cdot RT_{TR} \qquad \textit{Eqn. 31-3}$$

after the start of the transmission of the start bit.

For an frame with *n* payload bits, the receiver samples the RS8 sample of the stop bit

$$rx_{STOP} = (n + 1) \cdot 16 \cdot RT_{RE} + 7 \cdot RT_{RE} \qquad \textit{Eqn. 31-4}$$

after the successful qualification of the start bit.

To ensure error-free reception of the stop bit, the transmitter must start the transmission of the stop bit before the receiver samples RS8.

$$tx_{STOP} < rx_{STOP} \qquad \textit{Eqn. 31-5}$$

The maximum percent difference between the receiver baud rate and the transmitter baud rate is:

$$\Delta baudrate \leq \left(\frac{rx_{STOP} - tx_{STOP}}{rx_{STOP}}\right) \times 100 \qquad \textit{Eqn. 31-6}$$

The maximum percent differences for the supported frames is given in Table 31-20

Table 31-20. Faster Receiver Maximum Tolerance

Payload Bits	Max Baudrate Difference	tx_{STOP}	rx_{STOP}
8	4.63%	144	151
9	4.19%	160	167
13	3.03%	224	231

31.4.4.2 Slower Receiver Tolerance

In this case, the receiver has a slower baud rate than the transmitter, thus the stop bit sampling is still running while the next start bit is already transmitted. To ensure the correct noise- and framing error-free reception of the stop bit, the samples RS8, RS9, and RS10 must be located in the transmitted stop bit as shown in Figure 31-24.

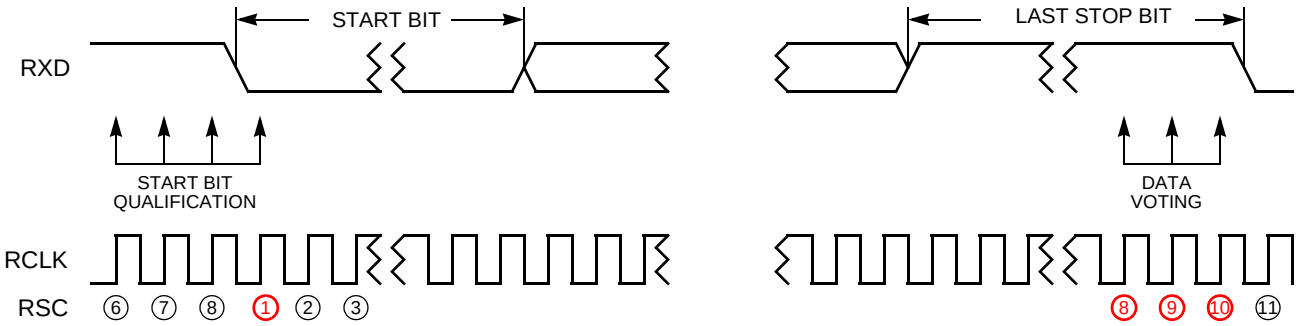


Figure 31-24. Slower Receiver

The maximum tolerance that ensures error-free reception can be calculated with the assumption that RS11 is sampled in the transmitted start bit and RS10 is sampled in the last stop bit.

For an frame with *n* payload bits and *s* stop bits, the transmitter starts the transmission of the next start bit

$$tx_{START} = (n + s + 1) \cdot 16 \cdot RT_{TR} \qquad \textit{Eqn. 31-7}$$

after the start of the transmission of the previous start bit.

For an frame with *n* payload bits and *s* stop bits, the receiver samples the RS10 sample of the last stop bit

$$rx_{STOP} = (n + s) \cdot 16 \cdot RT_{RE} + 9 \cdot RT_{RE} \qquad \textit{Eqn. 31-8}$$

after the successful qualification of the start bit.

To ensure error-free reception of the last stop bit, the transmitter must start the transmission of the start bit after the receiver samples RS10.

$$rx_{STOP} < tx_{START} \qquad \textit{Eqn. 31-9}$$

The maximum percent difference between the receiver baud rate and the transmitter baud rate is:

$$\Delta baudrate \leq \left(\frac{tx_{START} - rx_{STOP}}{tx_{START}} \right) \times 100 \qquad \textit{Eqn. 31-10}$$

The maximum percent differences for the supported frames is given in Table 31-21

Table 31-21. Slower Receiver Maximum Tolerance

Payload Bits	Stop Bits	Max Baudrate Difference	rx_{STOP}	tx_{START}
8	1	4.37%	153	160
9	1	3.97%	169	176
9	2	3.57%	185	196
13	2	2.73%	249	256

31.4.5 SCI Mode

31.4.5.1 SCI Mode Configuration

The application must configure the following bits and fields in order to achieve correct SCI operation.

- enable *SCI* Mode
  - eSCI LIN Control Register 1 (eSCI_LCR1)[LIN] = 0
- select *baud rate*
  - eSCI Baud Rate Register (eSCI_BRR)[SBR]
- select *receiver input* mode
  - eSCI Control Register 1 (eSCI_CR1)[LOOPS]
  - eSCI Control Register 1 (eSCI_CR1)[RSRC]
- select *frame format*
  - eSCI Control Register 1 (eSCI_CR1)[M]
  - eSCI Control Register 1 (eSCI_CR1)[PE]
  - eSCI Control Register 1 (eSCI_CR1)[WAKE]
  - eSCI Control Register 3 (eSCI_CR3)[M2]
- select *parity type*
  - eSCI Control Register 1 (eSCI_CR1)[PT]

31.4.5.2 Transmitter

The transmitter supports the transmission of all frame types defined in Table 31-15, of all break characters defined in Table 31-17, and of all idle characters defined in Table 31-18.

31.4.5.2.1 Transmitter States and Transitions

The transmitter has four basic states that are shown and described in Table 31-22. The state transitions that can triggered by the application commands are shown in Table 31-23. The state transitions that can

triggered by the module are shown in Table 31-24. The state diagram of the transmitter is shown in Figure 31-25.

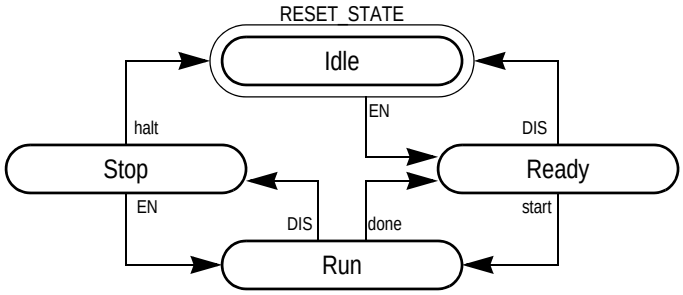


Figure 31-25. Transmitter State Diagram

The current state of the transmitter can be determined by the TE control bit in the eSCI Control Register 1 (eSCI_CR1) and the TACT status bit in eSCI Interrupt Flag and Status Register 1 (eSCI_IFSR1).

Table 31-22. Transmitter States

State	Indication		Description
	eSCI_CR1[TE]	eSCI_IFSR1[TACT]	
Idle	0	0	Transmitter is *disabled* and *no* transmission is running
Ready	1	0	Transmitter is *enabled* and *no* transmission is running
Run	1	1	Transmitter is *enabled* and transmission is running
Stop	0	1	Transmitter is *disabled* and transmission is running

The application triggers a transition described in Table 31-23 when it issues a command by writing to the TE bit in the eSCI Control Register 1 (eSCI_CR1). The transition is triggered only if the conditions are fulfilled. As a result of the transition the state of the transmitter is changed as shown in Figure 31-25 and the action given in Table 31-23 is executed.

Table 31-23. Transmitter Application Transitions

Transition	Command	Condition	Action	Description
EN	eSCI_CR1[TE] = 1	eSCI_CR1[TE] = 0	iPRE:= 1	Transmitter is *enabled* by application command.
DIS	eSCI_CR1[TE]:= 0	eSCI_CR1[TE] = 1		Transmitter is *disabled* by application command

The module transition shown in Table 31-24 are triggered when the described condition or event occurs. The send break bit SBK in the eSCI Control Register 1 (eSCI_CR1) is checked for the start condition. The internal commit bit iCMT, the transmitter active bit TACT in the eSCI Interrupt Flag and Status Register 1 (eSCI_IFSR1), the TDRE, and the TC flag in the eSCI Interrupt Flag and Status Register 1 (eSCI_IFSR1) are changed as a action result of the transition.

Table 31-24. Transmitter Module Transitions

Transition	Condition	Action	Description
start	(State = Ready) and (SBK = 1 or iPRE = 1 or iCMT = 1)	TACT = 1	Start of transmission of data frame or special character when data are available or character transmission request is pending.
done	State = Run and last stop bit transmitted	TACT = 0 TC = (SBK & iPRE & TC)	Finished transmission of data frame or special character and transmitter still enabled. Transmission is complete if no transmit request is pending.
halt	State = Stop and last stop bit transmitted	TACT = 0 TC = 1 iCMT = 0	Finished transmission of data frame or special character and transmitter was disabled.

31.4.5.2.2 Frame and Character Transmission

The transmitter starts the transmission of a data frame or special character when the condition for the *start* transition as described in Table 31-24 is fulfilled. There are three source for data or character transmission. The priority among these source are specified in Table 31-25. All three sources can be available at one point in time.

Table 31-25. Transmit Source Priority

Priority	Indication	Transmission Source
(highest) 0	eSCI_CR1[SBK] = 1	Break character.
1	iPRE = 1	Preamble.
(lowest) 2	iCMT = 1	eSCI SCI Data Register (eSCI_SDR) frame.

31.4.5.2.3 CPU Controlled SCI Data Frame Transmission

The transmission of a data frame is started when the transmitter is in its ready state and only the commit bit iCMT is set.

As the first step, the content of the eSCI SCI Data Register (eSCI_SDR) is transferred into the internal transmitter shift register. When this transfer is finished, the internal commit bit iCMT is cleared and the transmit data register empty flag TDRE in the eSCI Interrupt Flag and Status Register 1 (eSCI_IFSR1) is set. If the transmit interrupt enable bit TIE in the eSCI Control Register 1 (eSCI_CR1) is also set, the TDRE flag generates a transmitter interrupt request.

The transmitter shift register then shifts a frame out through the TXD output signal, which is prefaced with a start bit and appended with the parity bit, if configured, and the configured number of stop bits.

When the last stop bit has been transmitted and the application has not disabled the transmitter, the transmitter returns to the ready state via the *done* transition. If no frame or character transmit request is pending, the transfer complete flag TC in the eSCI Interrupt Flag and Status Register 1 (eSCI_IFSR1) is set.

If the application has disabled the transmitter while the frame is transmitted and stop bit has been transmitted, the transmitter goes into the idle state via the *halt* transition. The transfer complete flag TC in

the eSCI Interrupt Flag and Status Register 1 (eSCI_IFSR1) is set and the internal commit bit iCMT is cleared.

31.4.5.2.4 DMA Controlled SCI Data Frame Transmission

In this mode, the eSCI module handles the generation of data frames internally.

When new data required for transmission, the module generates the transmit DMA request and the DMA controller delivers the required data via write accesses to the eSCI SCI Data Register (eSCI_SDR). The write access to the low byte of the eSCI SCI Data Register (eSCI_SDR) triggers the transmission of the data. The write access to the high byte of the eSCI SCI Data Register (eSCI_SDR) triggers no internal operation.

The application requests the eSCI module to enter this mode by setting the TXDMA bit in the eSCI Control Register 2 (eSCI_CR2). From this point in time, the module starts the generation of DMA requests and frame transmission. Before entering this mode, the application should perform the following actions:

1. Configure the module for SCI mode.
2. Enable the transmitter by setting TE in the eSCI Control Register 1 (eSCI_CR1) to 1.
3. Set up the DMA controller channel and provide frame data in system memory.

Figure 31-26 shows an overview of the DMA-controlled date frame transmission.

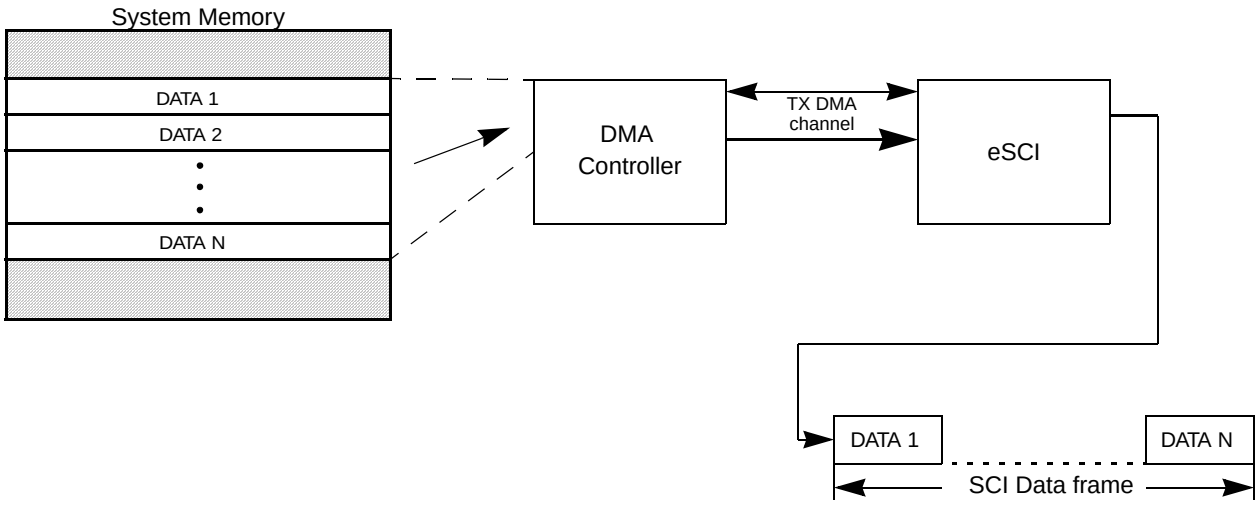


Figure 31-26. DMA Controlled SCI Data Frame Generation

NOTE

A received SCI frame is not written into the SCI Data Registers and the Overrun (OR) flag is not set in the SCI Status Register 1 (SCISR1), if:

- The eSCI has received the last data bit of an SCI frame n,
- The Receive Data Register Full (RDRF) flag is still set in the SCISR1 after the reception of SCI frame n-1, and

- During the reception of the STOP bit of frame n the host reads the SCI Data Registers, and clears the RDRF flag

In this case the RDRF flag is erroneously set again by the controller instead of the OR flag. Thus, the host reads the data of frame n-1 a second time, and the data of frame n is lost.

The application should ensure that the data of the foregoing frame is read out from the SCI Data Registers before the last data bit of the actual frame is received.

31.4.5.2.5 Parity Generation

The eSCI module generates the parity bit in transmitted data frame when the parity enable bit PE in the eSCI Control Register 1 (eSCI_CR1) is set. The parity type bit PT in the eSCI Control Register 1 (eSCI_CR1) defines whether the odd or even parity is generated.

31.4.5.2.6 Preamble Transmission

The transmission of a preamble is started when the transmitter is in ready state, the internal iPRE bit, which is not visible to the application, is set, and the SBK in the eSCI Control Register 1 (eSCI_CR1) is clear.

After the transmission of the stop bit and if the application has not disabled the transmitter, the transmitter returns to the ready state via the *done* transition. If no frame or character transmit request is pending, the transfer complete flag TC in the eSCI Interrupt Flag and Status Register 1 (eSCI_IFSR1) is set.

If the application has disabled the transmitter while the preamble is transmitted and if the stop bit has been transmitted, the transmitter goes into the idle state via the halt transition. The transfer complete flag TC in the eSCI Interrupt Flag and Status Register 1 (eSCI_IFSR1) is set and the internal commit bit iCMT is cleared.

31.4.5.2.7 Break Character Transmission

The transmission of a break character is started when the transmitter is in ready state and the send break character bit SBK in the eSCI Control Register 1 (eSCI_CR1) is set. After the transmission of the break character and if the application has not disabled the transmitter, the transmitter returns to the ready state via the *done* transition and restarts the transmission. As long as SBK bit remains set, the transmitter continues to send break characters.

When the application has cleared the SBK bit or has disabled the transmitter, the transmitter continues to transmit the current break character. After it has finished the transmission of this break character it transmits a stop bit. The stop bit at the end of a break character sequence guarantees the recognition of the start bit of the next data frame.

After the transmission of the stop bit and if the application has not disabled the transmitter, the transmitter returns to the ready state via the *done* transition. If no frame or character transmit request is pending, the transfer complete flag TC in the eSCI Interrupt Flag and Status Register 1 (eSCI_IFSR1) is set.

If the application has disabled the transmitter while the break character is transmitted and if the stop bit has been transmitted, the transmitter goes into the idle state via the *halt* transition. The transfer complete

flag TC in the eSCI Interrupt Flag and Status Register 1 (eSCI_IFSR1) is set and the internal commit bit iCMT is cleared.

31.4.5.3 Receiver

The receiver supports the reception of all data frame types defined in Table 31-15 and Table 31-16, of all break characters defined in Table 31-17, and of all idle characters defined in Table 31-18.

31.4.5.3.1 Receiver States and Transitions

The receiver has four basic states that are shown and described in Table 31-26. The state transitions that can triggered by the application commands are shown in Table 31-27. The state transitions that can triggered by the module are shown in Table 31-28. The state diagram of the transmitter is shown in Figure 31-27.

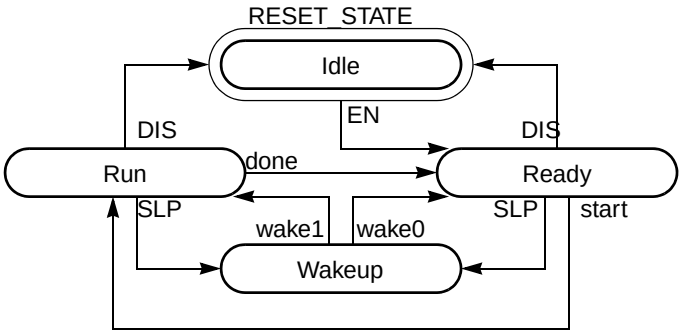


Figure 31-27. Receiver State Diagram

The current state of the receiver can be determined by the RE and RWU bit in the eSCI Control Register 1 (eSCI_CR1) and the RACT status bit in eSCI Interrupt Flag and Status Register 1 (eSCI_IFSR1).

Table 31-26. Receiver States

State	Indication			Description
	RE	RACT	RWU	
Idle	0	0	0	Receiver is *disabled* and *no* reception is running
Ready	1	0	0	Receiver is *enabled* and *no* reception is running
Run	1	1	0	Receiver is *enabled* and reception is running
Wakeup	1	—	1	Receiver is in wakeup mode

The application triggers a transition described in Table 31-27 when it issues a command by writing to the RE bit in the eSCI Control Register 1 (eSCI_CR1). The transition is triggered only if the conditions are fulfilled. As a result of the transition the state of the receiver is changed as shown in Figure 31-27 and the action given in Table 31-27 is executed.

Table 31-27. Receiver Application Transition

Transition	Command	Condition	Action	Description
EN	RE = 1	RE = 0		Receiver is *enabled* by application command.
DIS	RE = 0	RE = 1		Receiver is *disabled* by application command
SLP	RWU = 1	RE = 1		Receiver is set into wakeup mode

The module transition shown in Table 31-28 are triggered when the described event occurs.

Table 31-28. Receiver Module Transition

Transition	Condition	Action	Description
start	(State = Ready, Run) and (start bit received)	RACT:= 1	Start of reception of data frame or break character.
done	(State = Run) and (idle character received)	RACT:= 0	Idle Character received.
wake0	(State = Wakeup) and (idle character received)	RWU:= 0	Wakeup Idle Character received.
wake1	(State = Wakeup) and (address frame received)	RWU:= 0	Wakeup address frame received.

31.4.5.3.2 Receiver Input Mode Selection

This section describes the three receiver input modes supported by the eSCI module. The modes are selected by the LOOPS and RSRC control bits in the eSCI Control Register 1 (eSCI_CR1).

31.4.5.3.3 Dual Wire Mode

In dual wire mode, the eSCI uses the TXD pin for transmitting and the RXD pin for data receiving.

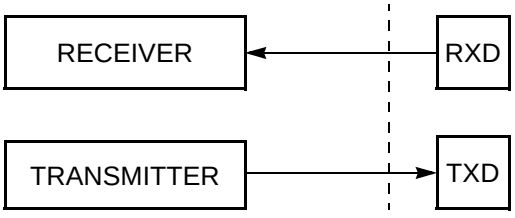


Figure 31-28. Dual Wire Mode

31.4.5.3.4 Single Wire Mode

In single wire mode, the RXD pin is disconnected from the eSCI module and the TXD pin is used for both receiving and transmitting.

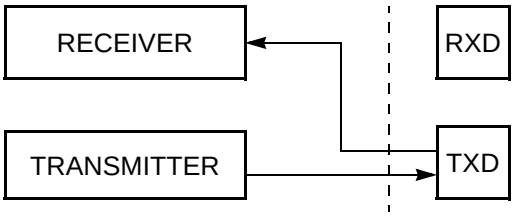


Figure 31-29. Single Wire Mode

The TXDIR bit (eSCI_CR2[1]) determines whether the TXD pin is going to be used as an input (TXDIR = 0) or an output (TXDIR = 1) in this mode of operation.

31.4.5.3.5 Loop Mode

In loop mode, the input of the receiver is driven by the output of the transmitter. The RXD pin is disconnected from the eSCI module.

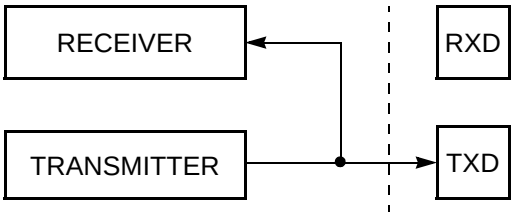


Figure 31-30. Loop Mode

31.4.5.3.6 Frame and Character Reception

The receiver is started when it is in ready or wakeup state and on the selected receiver input (see Section 31.4.5.3.2, Receiver Input Mode Selection) an active signal is sampled. The receiver enters the run or wakeup state. The received bits are recovered by the bit sampling described in Section 31.4.5.3.13, Bit Sampling. During the reception, the received bits are shifted into the internal shift register.

31.4.5.3.7 Break Character Detection

The receiver does not provide any means to detect the reception of a break character. Instead, break characters are processed as data frames. Due to the received 0 at the stop bit location, the reception of a break character causes at least a framing error. The error reporting is performed as described in Section 31.4.5.4, Reception Error Reporting.

31.4.5.3.8 Idle Character Detection

The start point of the idle character detection is controlled by the idle line type bit ILT in the eSCI Control Register 1 (eSCI_CR1).

If the ILT bit is 0, the idle character detection starts always immediately after the reception of a bit with the value 0. In this mode, a data frame with a payload section of all ones is erroneously detected as an idle character.

If the ILT bit is 1, the idle character detection starts after the reception of the last stop bit.

31.4.5.3.9 CPU Controlled SCI Data Frame Reception

This section describes the reception process when the receiver is in the run state.

When the required number of frame bits have been received, the payload bits of the received frame are transferred into eSCI SCI Data Register (eSCI_SDR) if the RDRF flag is 0. The receive data register full flag RDRF in eSCI Interrupt Flag and Status Register 1 (eSCI_IFSR1) is set. If the receive interrupt enable bit RIE in the eSCI Interrupt Flag and Status Register 1 (eSCI_IFSR1) is set, the RDRF interrupt request is generated.

If an idle character has been detected, the IDLE flag in the eSCI Interrupt Flag and Status Register 1 (eSCI_IFSR1) is set. If the idle line interrupt enable bit ILIE in the eSCI Control Register 1 (eSCI_CR1) is set, the IDLE interrupt request is generated.

If any of the receiver errors described in Section 31.4.5.4, Reception Error Reporting, have been occurred, that corresponding flags are set.

If the application disabled the receiver by clearing the receiver enable bit RE in the eSCI Interrupt Flag and Status Register 1 (eSCI_IFSR1), the current frame is discarded and no flags are updated.

31.4.5.3.10 DMA Controlled SCI Data Frames Reception

In this mode, the eSCI module controls the reception of SCI data frames automatically and utilizes the connected DMA channels. Figure 31-31 shows an overview of the DMA-controlled SCI data frame reception. The RX DMA channel is used to transfer the received frame data into the memory.

When new data is received, the module generates the receive DMA request and the DMA controller retrieves the provided data from the eSCI SCI Data Register (eSCI_SDR). The read access from the low byte of the eSCI SCI Data Register (eSCI_SDR) signals the end of the DMA cycle for the current data and triggers the reception of new data. The read access from the eSCI SCI Data Register (eSCI_SDR) triggers no internal action

The application request the eSCI module to enter this mode by setting the RXDMA bit in the eSCI Control Register 2 (eSCI_CR2). From this point in time, the module start the generation of DMA requests and frame transmission and reception. Before entering this mode, the application should perform the following actions:

1. Configure the module for SCI mode.
2. Enable the receiver by setting RE in eSCI Control Register 1 (eSCI_CR1) to 1.
3. Set up the DMA controller channel.

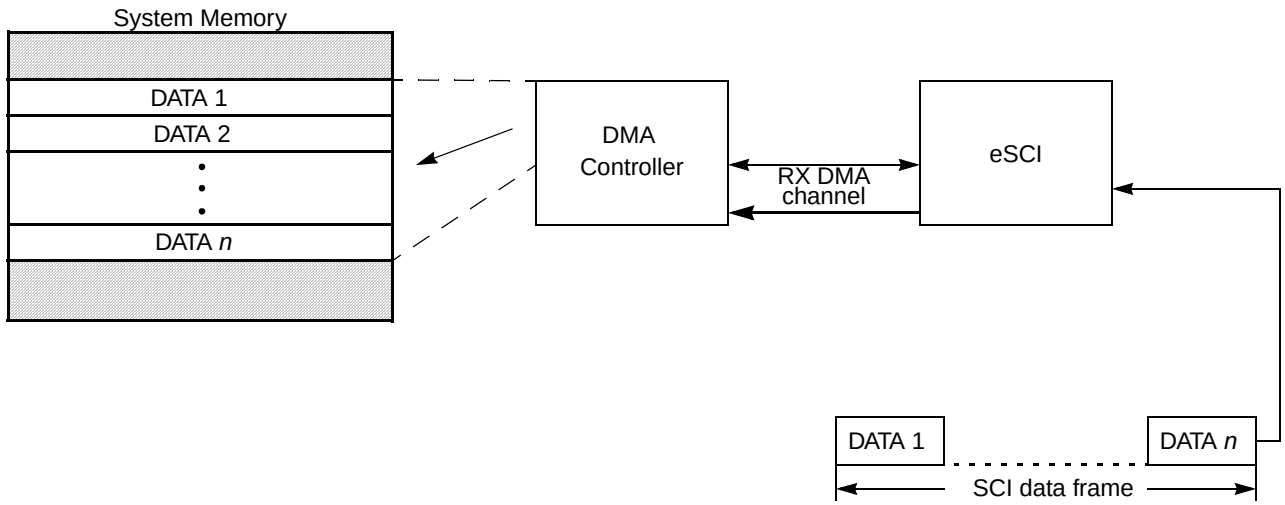


Figure 31-31. DMA-Controlled SCI Data Frame Reception

31.4.5.3.11 Receiver Overrun

When the eSCI module has received a frame and attempts to transfer the payload data of the received frame into the eSCI SCI Data Register (eSCI_SDR) but neither the application nor the DMA controller has read the eSCI SCI Data Register (eSCI_SDR) since its last update, the overrun flag OR in the eSCI Interrupt Flag and Status Register 1 (eSCI_IFSR1) is set. The data contained in eSCI SCI Data Register (eSCI_SDR) are not changed and the received data are lost.

31.4.5.3.12 Wake-up Frame Reception

This section describes the reception process when the receiver is in the wakeup state.

When the required number of frame bits have been received, the payload bits of the received frame are transferred into eSCI SCI Data Register (eSCI_SDR) if the RDRF flag is 0.

If the *address-mark* wake-up mode is selected and the received frame has the address bit set, the receive data register full flag RDRF in eSCI Interrupt Flag and Status Register 1 (eSCI_IFSR1) is set. If the receive interrupt enable bit RIE in the eSCI Interrupt Flag and Status Register 1 (eSCI_IFSR1) is set, the RDRF interrupt request is generated. The RWU bit is cleared, and the receiver enters the run state via the wake1 transition.

If the *idle line* wake-up mode is selected and the receiver has detected an idle character, The RWU bit is cleared, and the receiver enters the ready state via the wake0 transition.

If any of the receiver errors described in Section 31.4.5.4, Reception Error Reporting, have been occurred, that corresponding flags are set.

31.4.5.3.13 Bit Sampling

The receiver samples the selected receiver input (see Section 31.4.5.3.2, Receiver Input Mode Selection) with the receiver clock RCLK. The sampling for start bit detection is shown in Figure 31-32. The sampling for data and stop bit reception is shown in Figure 31-33. The samples indicated by dashed arrows are not

used by the receiver. The received data bits are transferred into the internal shift register after the data strobing. If noise or framing errors are detected, this is flagged as described in Section 31.4.5.4, Reception Error Reporting.

31.4.5.3.14 Bit Synchronization

To adjust for baud rate mismatch, a synchronization of the cyclic sample counter RSC is performed during start bit reception as described in Section 31.4.5.3.15, Start Bit Sampling.

Additionally, the synchronization of the cyclic sample counter RSC can be configured to be performed during data bit reception as described in Section 31.4.5.3.16, Data Bit Sampling.

31.4.5.3.15 Start Bit Sampling

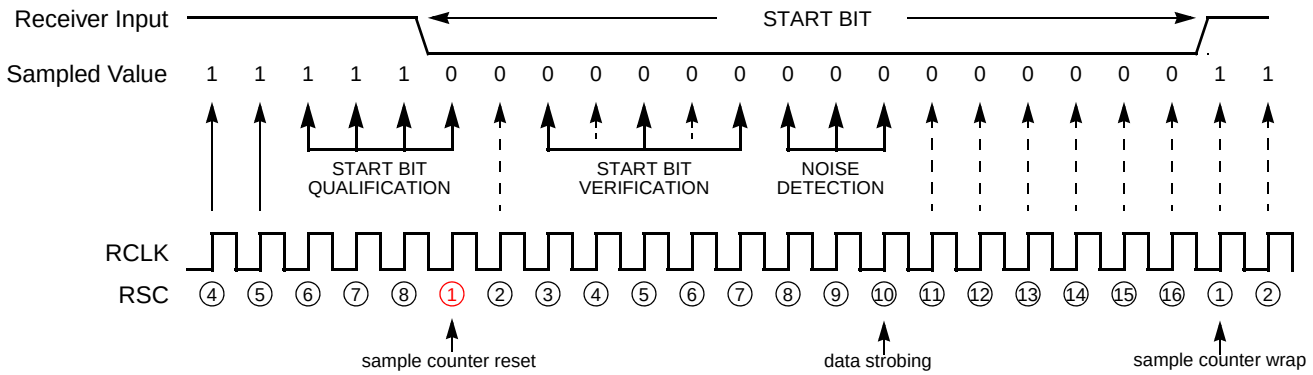


Figure 31-32. Start Bit Sampling and Strobing

Start Bit Qualification

To adjust for baud rate mismatch, the cyclic sample counter RSC is re-synchronized by reset after successful start bit qualification. A start bit is successfully qualified, if no reception is ongoing and three consecutive high samples are followed immediately by a low sample.

Start Bit Verification

After the successful start bit qualification the receiver starts to verify the start bit by a two out of three samples majority voting.

A start bit is verified if at least two out of the three sample RSC3, RSC5, and RSC5 are sampled low. Noise is detected when exactly one out of the three samples is high. The results of the start bit verification is summarized in Table 31-29.

Table 31-29. Start Bit Verification Result

[RS3, RS5, RS7]	Start Bit Verified	Noise Detected
000	Yes	No
001	Yes	Yes
010	Yes	Yes

Table 31-29. Start Bit Verification (continued)Result

[RS3, RS5, RS7]	Start Bit Verified	Noise Detected
100	Yes	Yes
011	No	No
101	No	No
110	No	No
111	No	No

If the start bit verification was *not successful*, the receiver resumes the start bit qualification. If the start bit verification was *successful*, the receiver continues sampling to perform noise detection on the samples at RS8, RS9, and RS10. The results of the start bit noise detection is summarized in Table 31-30.

Table 31-30. Start Bit Noise Detection

[RS8, RS9, RS10]	Noise Detected
000	No
001	Yes
010	Yes
100	Yes
011	Yes
101	Yes
110	Yes
111	Yes

31.4.5.3.16 Data Bit Sampling

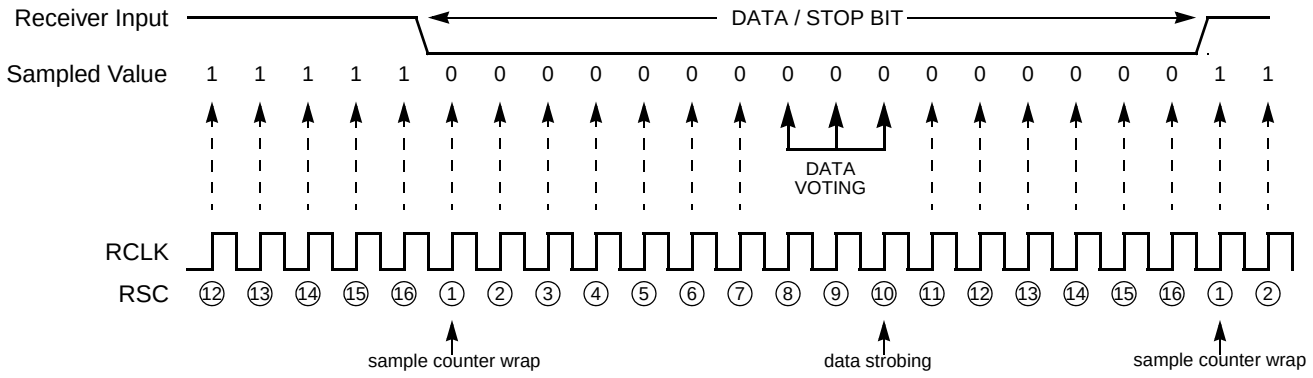


Figure 31-33. Data and Stop Bit Sampling and Strobing

To determine the value of a data bit and to detect noise, a two out of three majority voting is performed on the samples RS8, RS9, and RS10. Table 31-31 summarizes the results of the data bit sample. The receiver detects the number of data bit according to the selected frame format.

Table 31-31. Data Bit Sampling

[RS8, RS9, RS10]	Data Bit Value	Noise Detected
000	0	No
001	0	Yes
010	0	Yes
100	0	Yes
011	1	Yes
101	1	Yes
110	1	Yes
111	1	No

31.4.5.3.17 Data Bit Synchronization

To adjust for baud rate mismatch during the reception of data bits, the cyclic sample counter RSC can be configured to be synchronized on falling edges during data bit reception. This kind of synchronization is performed only if the synchronization mode bit SYNM in the eSCI Control Register 3 (eSCI_CR3) is 0.

Data Bit Synchronization (Right Shifted Edges)

This kind of sample counter synchronization happens if the transmitter is slower than the receiver. The reset behavior of the sample counter is shown in Figure 31-34. The sample counter reset condition is:

1. The data bit N-1 is sampled as 1, and
2. the data bit N is sampled as 0, and
3. a falling edge consisting of three consecutive 1-samples and a following 0-sample is detected, and
4. the 0-sample of the falling edge is received at data bit N sample j, with 1 <= j <= 8.

If the condition is fulfilled, the sample counter is reset 16 RCLK cycles after the 0 of the falling edge condition was received. The bit counter is not increased.

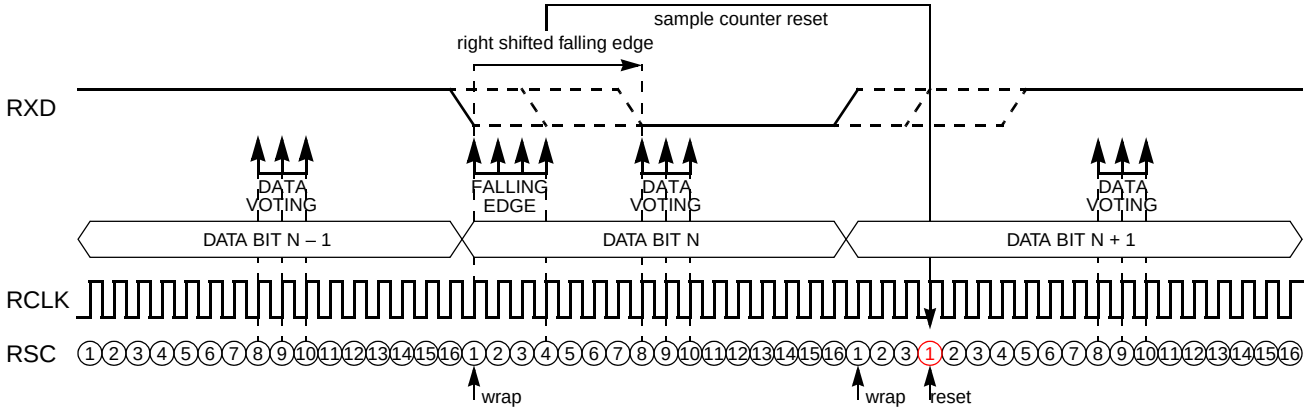


Figure 31-34. Data Bit Synchronization (Right Shifted Edges)

Data Bit Synchronization (Left Shifted Edges)

This kind of sample counter synchronization happens if the transmitter is faster than the receiver. The reset behavior of the sample counter is shown in Figure 31-35. The sample counter reset condition is:

1. The data bit N – 1 is sampled as 1, and
2. The data bit N is sampled as 0, and
3. A falling edge consisting of three consecutive 1-samples and a following 0-sample is detected, and
4. The 0-sample of the falling edge is received at data bit N sample j, with $11 \le j \le 16$.

If the condition is fulfilled, the sample counter is reset 16 RCLK cycles after the 0-sample of the falling edge condition was received. The bit counter is increased by 1.

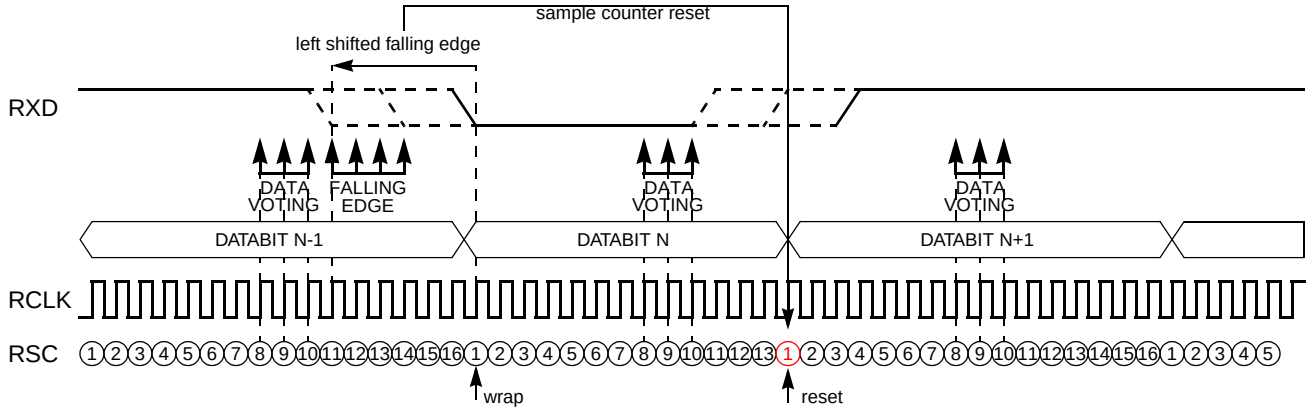


Figure 31-35. Data Bit Synchronization (Left Shifted Edges)

If the 0-sample of the falling edge condition is received at sample 9 or 10, no sample counter synchronization is performed.

31.4.5.3.18 Stop Bit Verification

The reception of a valid stop bit is verified if at least two out of the sample RS8, RS9, and RS10 are sampled high. If this is not that case, a framing error is detected. Noise is detected if not all of the samples are of the same value. The results of the stop bit verification are summarized in Table 31-32.

Table 31-32. Stop Bit Verification

[RS8, RS9, RS10]	Stop Bit Verified	Framing Error Detected	Noise Detected
000	No	Yes	No
001	No	Yes	Yes
010	No	Yes	Yes
100	No	Yes	Yes
011	Yes	No	Yes
101	Yes	No	Yes
110	Yes	No	Yes
111	Yes	No	No

31.4.5.3.19 Parity Checking

The eSCI module calculates the parity of a received character and checks is versus the received parity bit in the received data frame when the parity enable bit PE in the eSCI Control Register 1 (eSCI_CR1) is set. The parity type bit PT in the eSCI Control Register 1 (eSCI_CR1) defines whether to check for odd or even parity is generated. If an parity error is detected, this is reported as described in Section 31.4.5.4, Reception Error Reporting.

31.4.5.4 Reception Error Reporting

The receiver can detect four error types: parity errors, framing errors, noise errors, and the overrun error.

The receiver reports the errors detected during frame reception at the end of the reception of the last stop bit of a frame. For error reporting the receiver utilizes the OR, NF, FE, and PF flags in the eSCI Interrupt Flag and Status Register 1 (eSCI_IFSR1).

If the receiver has detected an overrun as described in Section 31.4.5.3.11, Receiver Overrun, only the OR flag is set. All other error flags are not updated.

If the receiver has *not* detected an overrun and has detected noise as described in Section 31.4.5.3.13, Bit Sampling, the NF flag is set.

If the receiver has *not* detected an overrun and has detected a framing error as described in Section 31.4.5.3.13, Bit Sampling, the FE flag is set.

If the receiver has *not* detected an overrun and has detected a parity error as described in Section 31.4.5.3.19, Parity Checking, the PF flag is set.

31.4.5.5 Multiprocessor Communication

The multiprocessor communication allows one processor to send blocks of frames to other processors on the same serial link. To avoid the received data interrupt for frames not intended for the processor, the eSCI receiver can be put into the wakeup state. If the receiver is in the wakeup state, the eSCI still loads the received data into the eSCI SCI Data Register (eSCI_SDR), but does not set the RDRF flag and consequently does not request the RDRF interrupt.

The receiver leaves the wakeup state and clears the RWU bit in the eSCI Control Register 1 (eSCI_CR1) when the wakeup pattern configured by WAKE bit in eSCI Control Register 1 (eSCI_CR1) is received. The eSCI module supports two types of wakeup patterns, the idle-line wakeup pattern and the address-mark wakeup pattern.

31.4.5.5.1 Idle-Line Wakeup

The idle-line wakeup mode is selected when the WAKE bit in eSCI Control Register 1 (eSCI_CR1) is 0. In this mode, the receiver leaves the wakeup state, when an idle character is detected as described in Section 31.4.5.3.8, Idle Character Detection. The next received frame is the address frame, which contains address information that can be evaluated by the application. If the application decides not to receive the frame block, it can set the RWU bit in the eSCI Control Register 1 (eSCI_CR1) and return the receiver to the wakeup state.

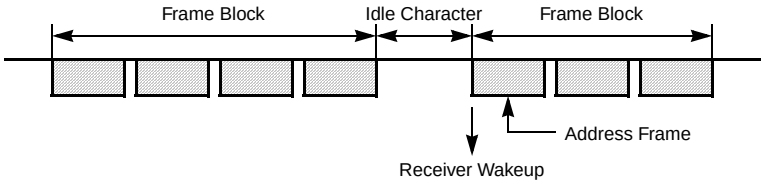


Figure 31-36. Idle-Line Wakeup Format

31.4.5.5.2 Address-Mark Wakeup

The address-mark wakeup mode is selected when the WAKE bit in eSCI Control Register 1 (eSCI_CR1) is 1. If the WAKE bit is set, the address bit is added to the frame format. In this mode, the receiver leaves the wakeup state, when a data frame with the address bit value of 1 was received. This frame is the address frame and contains address information that can be evaluated by the application. If the application decides not to receive the frame block, it can set the RWU bit in the eSCI Control Register 1 (eSCI_CR1) and return the receiver to the wakeup state. All data frames that belong to the frame block must have the address bit cleared.

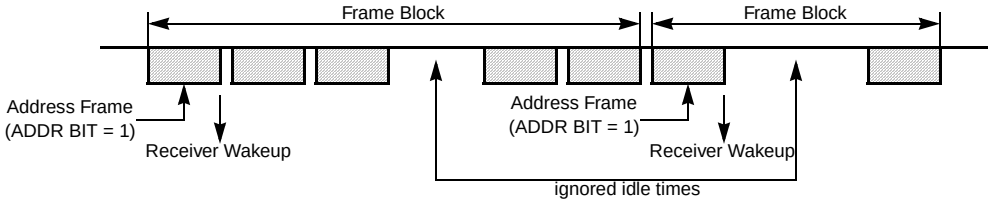


Figure 31-37. Address-Mark Wakeup Format

31.4.6 LIN Mode

The eSCI provides support for the LIN protocol. It can be used to automate most tasks of a LIN master. In conjunction with the DMA interface it is possible to transmit entire LIN frames and sequences of LIN frames as well as to receive data from LIN slaves without application intervention. There is no special support for LIN slave mode.

31.4.6.1 LIN Mode Configuration

The application must configure the following bits and fields in order to achieve correct LIN operation. The configuration of bits and fields not mentioned in this section depend on the connected LIN slaves and the current application.

- enable *LIN* Mode
  - eSCI LIN Control Register 1 (eSCI_LCR1)[LIN]:= 1
- select *RXD* pin as receiver input
  - eSCI Control Register 1 (eSCI_CR1)[LOOPS]:= 0
  - eSCI Control Register 1 (eSCI_CR1)[RSRC]:= 0
- select *LIN byte fields* as used frame format

  - eSCI Control Register 1 (eSCI_CR1)[M]:= 0
  - eSCI Control Register 1 (eSCI_CR1)[PE]:= 0
  - eSCI Control Register 1 (eSCI_CR1)[WAKE]:= 0
  - eSCI Control Register 3 (eSCI_CR3)[M2]:= 0
- select *break character length* of 13 bit as required by LIN 2.0
  - eSCI Control Register 2 (eSCI_CR2)[BRCL]:= 1
- select both transmitter and receiver *reset on bit error* detection
  - eSCI LIN Control Register 1 (eSCI_LCR1)[LDBG]:= 0
- select transmission stop on *bit error* detection
  - eSCI Control Register 2 (eSCI_CR2)[BESTP]:= 1
- select transmission DMA stop on *bit error* detection
  - eSCI Control Register 2 (eSCI_CR2)[BSTP]:= 1
- enable both *transmitter* and *receiver*
  - eSCI Control Register 1 (eSCI_CR1)[TE]:= 1
  - eSCI Control Register 1 (eSCI_CR1)[RE]:= 1

31.4.6.2 LIN Frame Formats

The term LIN frame refers to a sequence of LIN byte fields preceded by a break character, both are described in Section 31.4.2, Frame Formats. The eSCI module allows to generate LIN frames for LIN slaves of LIN standards 1.3 and 2.0.

31.4.6.2.1 Standard LIN Frames

A standard LIN frame, shown in Figure 31-38 consists of a break character, a sync field, an ID field, zero or more data fields, and a checksum field. The data fields and the checksum field are generated by the LIN master for TX LIN frames and generated by the LIN slave for RX LIN frames. The header fields are always generated by the LIN master.

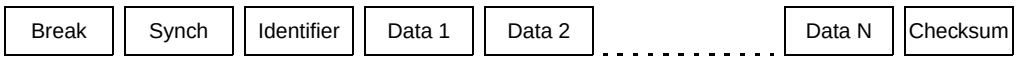


Figure 31-38. Standard LIN Frame Format

31.4.6.2.2 CRC Enhanced LIN Frames

The CRC Enhanced LIN frames shown in Figure 31-39 contain two additional CRC byte fields. These fields are located between the last data field and the Checksum field. The value of the CRC is calculated on the same byte fields as the Checksum is calculated on. The polynom used for the CRC calculation is defined by eSCI LIN CRC Polynomial Register (eSCI_LPR). The eSCI module generates the CRC fields for TX frames and checks the CRC fields for RX frames if the CRC bit in the eSCI LIN Transmit Register (eSCI_LTR) was written with a value of 1.

Figure 31-39. CRC Enhanced LIN Frame Format

The CRC Enhanced LIN frames are not part of the LIN standard.

31.4.6.3 LIN TX Frame Generation

The eSCI module supports two modes of LIN TX Frame generation. In the application controlled mode, the application provides the required frame configuration and frame data by subsequent write accesses to the eSCI LIN Transmit Register (eSCI_LTR). In the DMA generation mode, the DMA controller provides the required frame configuration and frame data in response to DMA requests generated by the eSCI module.

31.4.6.3.1 CPU Controlled LIN TX Frame Generation

In this mode, the application initiates the generation of a LIN TX Frame and provides the data to be transmitted by a sequence of subsequent CPU write accesses to the eSCI LIN Transmit Register (eSCI_LTR). When the eSCI module has processed the data written into the eSCI LIN Transmit Register (eSCI_LTR), the TXRDY interrupt flag in the eSCI Interrupt Flag and Status Register 2 (eSCI_IFSR2) is set.

The application should clear the TXRDY interrupt flag before writing data into the eSCI LIN Transmit Register (eSCI_LTR) because the eSCI module sets the TXRDY one clock cycle after the write access.

The first data written to the eSCI LIN Transmit Register (eSCI_LTR) provides the Identifier and Identifier Parity fields. The second data written defines the number of data bytes to be transmitted. The third data written defines the CRC and checksum generation. The TD bit has to be set to 1 in order to invoke the LIN TX frame generation. The value of the TO field is ignored by the eSCI module for LIN TX frames.

After the third data is written, the generation of a LIN TX frame is started. First, a break field is transmitted, then the synch field and the protected identifier field.

All subsequent write accesses to the eSCI LIN Transmit Register (eSCI_LTR) provide data bytes to be transmitted via the LIN bus. A data byte field is transmitted as soon as data are available. After the last data byte (defined by the value written to the LEN field) is sent, the configured CRC and checksum fields are sent out.

After the transmission of the checksum field of the LIN TX frame, the write access counter for the eSCI LIN Transmit Register (eSCI_LTR) is reset and the FRC interrupt flag in the eSCI Interrupt Flag and Status Register 2 (eSCI_IFSR2) is set.

NOTE

If the eSCI module enters stop while the transmit DMA is enabled and messages are being transmitted, when the CPU exits stop or doze mode, it is possible that DMA requests will not be generated by the eSCI module. To avoid this, the application should ensure that the eSCI module is idle before entering the stop mode. The eSCI module is idle when both transmitter and receiver active status bits in the Interrupt Flag and Status Register 1 (eSCI_IFSR1) are not set. The application should not trigger a new transmission on the eSCI module if the application is preparing for the stop mode.

31.4.6.3.2 DMA Controlled LIN TX Frame Generation

In this mode, the eSCI module handles the generation of an LIN TX Frame internally. When new data is ready for transmission, the module generates the transmit DMA request and the DMA controller delivers the required data. The application requests the eSCI module to enter this mode by setting the TXDMA bit in the eSCI Control Register 2 (eSCI_CR2). From this point in time, the module starts the generation of DMA requests and frame transmission. Before entering this mode, the application should perform the following actions:

1. Configure the module for LIN mode.
2. Enable the transmitter by setting TE in eSCI Control Register 1 (eSCI_CR1) to 1.
3. Set up the DMA controller channel and provide frame data in system memory.

Figure 31-40 shows an overview of the DMA-controlled LIN TX frame. The content of the fields in the memory is the same as described in eSCI LIN Transmit Register (eSCI_LTR) — LIN TX Frame Generation.

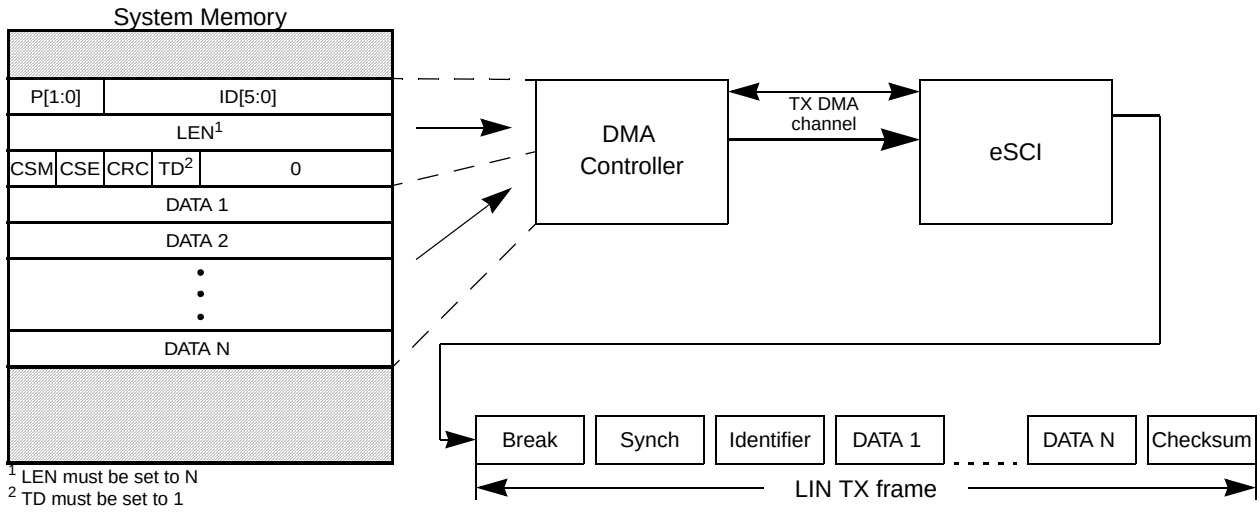


Figure 31-40. DMA Controlled LIN TX Frame Generation

31.4.6.4 LIN RX Frame Generation

The eSCI module supports two modes of LIN RX Frame generation and reception, the CPU controlled mode and the DMA controlled mode. In the CPU controlled mode, the application provides the required data by subsequent CPU write accesses to the eSCI LIN Transmit Register (eSCI_LTR) and retrieves the received data by subsequent CPU read accesses to the eSCI LIN Receive Register (eSCI_LRR). In the DMA controlled mode, the DMA controller provides the required frame configuration data in response to DMA requests generated by the eSCI module and transfers the received frame data to the memory in response to DMA requests generated by the eSCI module.

31.4.6.4.1 Application Controlled LIN RX Frames Generation

In this mode, the application initiates the generation of an LIN RX Frame by a sequence of subsequent CPU write accesses to the eSCI LIN Transmit Register (eSCI_LTR). When the eSCI module has processed the data written into eSCI LIN Transmit Register (eSCI_LTR), the TXRDY interrupt flag in the eSCI Interrupt Flag and Status Register 2 (eSCI_IFSR2) is set.

The application must clear the TXRDY interrupt flag before writing data into the eSCI LIN Transmit Register (eSCI_LTR) because the eSCI module sets the TXRDY one clock cycle after the write access.

The first data written to the eSCI LIN Transmit Register (eSCI_LTR) provides the Identifier and Identifier Parity fields. The second data written defines the number of data bytes requested from the LIN slave. The third data written defines the CRC and checksum generation. The TD bit must be set to 0 to invoke the RX frame generation. The TO field defines the upper part of the timeout value. The fourth byte written defines the lower part of the timeout value.

After the fourth byte is written, the generation of a LIN RX frame is started. First, a break field is transmitted, then the synch field and the protected identifier field. After the transmission of the protected identifier, the eSCI module starts to receive the frame data transmitted by the LIN slave. When the module has received a complete byte field, the received data are transferred into the eSCI LIN Receive Register (eSCI_LRR) and the receive data ready flag RXRDY in the eSCI Interrupt Flag and Status Register 2 (eSCI_IFSR2) is set.

The application can retrieve the received data by subsequent read access from eSCI LIN Receive Register (eSCI_LRR) after checking the RXRDY flag. The application must clear the RXRDY flag immediately after reading the eSCI LIN Receive Register (eSCI_LRR).

After the reception of the configured number of data from the slave, the module starts the reception of the configured CRC and Checksum byte fields. These data are not transferred into the eSCI LIN Receive Register (eSCI_LRR). The CRC and Checksum checking is performed internally. Errors are reported as described in Section 31.4.6.5, LIN Error Reporting.

After the reception of the checksum field of the LIN RX frame, the FRC interrupt flag in the eSCI Interrupt Flag and Status Register 2 (eSCI_IFSR2) is set.

NOTE

When the eSCI module is in LIN mode and transmits the Header of an LIN RX frame, if the CPU requests Stop Mode, the eSCI module may not acknowledge the Stop Mode request and will stay in Normal Operating Mode (not in lower power stop mode).

The application should ensure that no LIN transmission is running before it requests Stop Mode by checking the Transmit Active and Receive Active status bits in the eSCI Interrupt Flag and Status Register (eSCI_IFSR1[TACT] and eSCI_IFSR1[RACT]).

31.4.6.4.2 DMA Controlled LIN RX Frames Generation

In this mode, the eSCI module controls the generation of LIN RX frame header and the reception of the frame data automatically and utilizes the two connected DMA channels. Figure 31-40 provides a block diagram that shows an overview of the DMA Controlled LIN RX Frame generation and reception. The content of the header fields in the memory is the same as described in eSCI LIN Transmit Register (eSCI_LTR) — LIN RX frame generation. The TX DMA channel is used the fetch the LIN RX frame header and control information. The RX DMA channel is used to transfer the received frame data into the memory.

When new data required for transmission, the module generates the transmit DMA request and the DMA controller delivers the required data. When new data was received, the module generates the receive DMA request and the DMA controller retrieves the provided data.

The application request the eSCI module to enter this mode by setting the RXDMA bit in the eSCI Control Register 2 (eSCI_CR2). From this point in time, the module start the generation of DMA requests and frame transmission and reception. Before entering this mode, the application should perform the following actions:

1. Configure the module for LIN mode.
2. Enable transmitter and receiver by setting TE and RE in eSCI Control Register 1 (eSCI_CR1) to 1.
3. Set up the two DMA controller channels and provide frame header data in system memory.

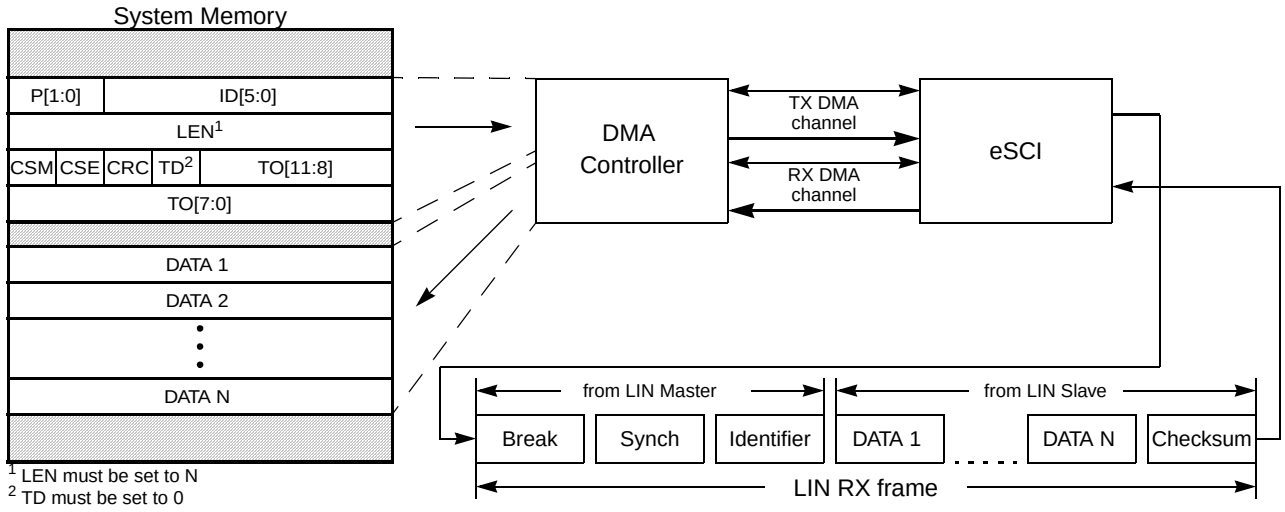


Figure 31-41. DMA Controlled LIN RX Frame Generation and Reception

31.4.6.5 LIN Error Reporting

This section describes error checking and the signaling of detected errors in LIN mode.

31.4.6.5.1 Physical Bus Error Detection

If the receiver input is sampled 0 for at least 31 sample clock cycles after the start of the transmission of a LIN frame, the physical bus error flag PBERR in the eSCI Interrupt Flag and Status Register 2 (eSCI_IFSR2) is set.

31.4.6.5.2 Unrequested Activity Detection

If an unrequested byte is received (i.e., a byte that is not part of an RX frame) that is not recognized as a wakeup or break character, the bit error flag BERR in the eSCI Interrupt Flag and Status Register 2 (eSCI_IFSR2) is set. In addition, the RXRDY flag is set. The LINRX register must be read before normal operations can proceed.

31.4.6.5.3 Standard Bit Error Detection

The standard bit error detection is performed on each byte field transmission.

During the transmission of the frame header and frame data, the receiver is running and receives the signal values on the serial bus. After the complete transmission of a byte field, the eSCI compares the data that was transmitted and the data that was received. If they do not match, the bit error flag BERR in the eSCI Interrupt Flag and Status Register 2 (eSCI_IFSR2) is set.

31.4.6.5.4 Fast Bit Error Detection

Fast Bit Error Detection has been designed to allow flagging of LIN bit errors while they occur, rather than flagging them after a byte transmission has completed (see Figure 31-42).

Figure 31-42. Fast Bit Error Detection on a LIN Bus

If fast bit error detection bit FBR in the eSCI Control Register 2 (eSCI_CR2) is set, the eSCI compares the transmitted and received data streams while the transmitter is active (not idle). Once a mismatch between the transmitted data and the received data is detected, the bit error flag BERR is set.

To adjust to different bus loads, the sample point at which the incoming bit is compared to the transmitted bit can be selected with the BESM bit in the eSCI Control Register 2 (eSCI_CR2). If eSCI_CR2[BESM] = 1, the comparison is performed with sample RS13. Otherwise, it is performed with RS9, as shown inFigure 31-43. See Section 31.4.5.3.13, Bit Sampling.

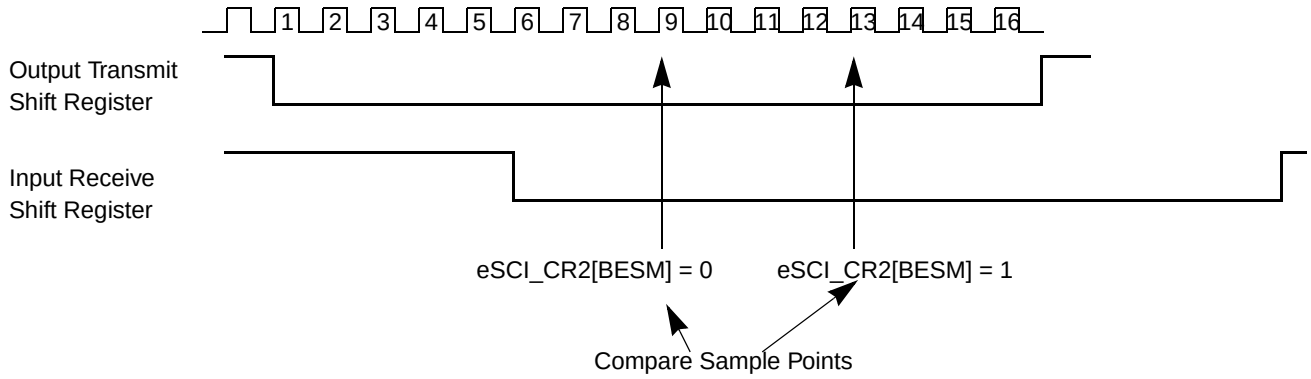


Figure 31-43. Timing Diagram Fast Bit Error Detection

NOTE

To calculate the exact position of the sample point with regard to the RX pin, the delays through the pads and the two Bus Clock cycle delay through the input synchronizer also needs to be taken into account.

31.4.6.5.5 Slave-Not-Responding-Error Detection

The Slave-Not-Responding-Error is defined in LIN Specification Package Revision 1.3; December 12, 2002; 6 ERROR AND EXCEPTION HANDLING. The LIN specification requires that a NO_RESPONSE_ERROR has to be detected if a message frame is not fully completed within the maximum length T_{FRAME_MAX} by any slave task upon transmission of the SYNCH and IDENTIFIER fields. The maximum frame length T_{FRAME_MAX} is defined in LIN Specification Package Revision 1.3; December 12, 2002; 3.3 LENGTH OF MESSAGE FRAME AND BUS SLEEP DETECT, as

$$T_{FRAME\_MAX} = (10 \cdot N_{DATA} + 45) \cdot 1.4 \qquad \textit{Eqn. 31-11}$$

where N_{DATA} is the number of data byte fields of the message frame.

The STO interrupt flag in the eSCI Interrupt Flag and Status Register 2 (eSCI_IFSR2) is set if a LIN RX frame was not fully received in the amount of time specified in the timeout value field TO in the eSCI LIN Transmit Register (eSCI_LTR). The time period starts with the falling edge of the transmitted LIN break character and is specified in units of transmit bits.

To achieve LIN compliant Slave-Not-Responding-Error detection, the timeout value TO in the eSCI LIN Transmit Register (eSCI_LTR) field has to be set to T_{FRAME_MAX} when a LIN RX frame is initiated.

31.4.6.5.6 Checksum Error Detection

If the checksum enable bit CSE in the eSCI LIN Transmit Register (eSCI_LTR) is set, checksum checking is performed based on the received checksum byte. The checksum mode is selected by the CSM bit in the eSCI LIN Transmit Register (eSCI_LTR). If the value received in the checksum bytes does not match the calculated checksum, the checksum error flag CKERR in the eSCI Interrupt Flag and Status Register 2 (eSCI_IFSR2) is set.

31.4.6.5.7 CRC Error Detection

CRC checking is performed on the two received CRC bytes CRC1 and CRC2 if the CRC Enhanced LIN frame format was selected by the CRC bit in the eSCI LIN Transmit Register (eSCI_LTR). If the value received in the two CRC bytes did not match the calculated CRC pattern, the CRC error flag CERR in the eSCI Interrupt Flag and Status Register 2 (eSCI_IFSR2) is set.

31.4.6.5.8 Overflow Detection

When the receiver has received the next byte field to be transferred into the eSCI LIN Receive Register (eSCI_LRR), but neither the application nor the RX DMA channel have read data from this register since the last update, the received data overflow flag OVFL in the eSCI Interrupt Flag and Status Register 2 (eSCI_IFSR2) is set. In this case, the content of the eSCI LIN Receive Register (eSCI_LRR) is not changed. The data received most recently are lost.

31.4.6.6 LIN Wakeup

The section describes the LIN wakeup behavior of the eSCI module.

31.4.6.6.1 LIN Wakeup Generation

The eSCI module can cause the LIN bus to exit the sleep mode by sending a break character. The application triggers the transmission of a break character by writing 1 to the LIN bus wakeup trigger WU in the eSCI LIN Control Register 1 (eSCI_LCR1). After the end of transmission of this break character, the transmitter neither sets the TXRDY flag in the eSCI Interrupt Flag and Status Register 2 (eSCI_IFSR2) nor starts the transmission of frame data until the wakeup delimiter period has expired. The wakeup delimiter period is defined by the WUD field in the eSCI LIN Control Register 1 (eSCI_LCR1).

To generate a valid wakeup character according to LIN 2.0, the eSCI first needs to be programmed to a baud rate lower than 32 kbaud, then WU can be set. Should the application require a higher baud rate, then this rate can be set once the wakeup character has been transmitted.

31.4.6.6.2 LIN Wakeup Reception

If the eSCI receives a valid wakeup condition on the selected receiver input, the LIN wakeup flag LWAKE in the eSCI Interrupt Flag and Status Register 2 (eSCI_IFSR2) is set. Since each valid wakeup condition violates the byte field structure, the frame error flag FE in the eSCI Interrupt Flag and Status Register 1 (eSCI_IFSR1) is also set.

The eSCI detects the following conditions as valid wakeup conditions:

- Reception of a break signal

- Reception of a LIN 1.x wakeup character (0x80, 0x00 or oxC0)
- Reception of a LIN 2.0 wakeup character (low pulse of 250 ms to 5 ms).
  To detect LIN 2.0 wakeup characters, the baud rate must to be set to 32 kbaud down to 1.6 kbaud.

NOTE

If the eSCI module is transmitting a LIN frame and the application sets and clears the LIN Finite State Machine Resync bit in the LIN Control Register 1 (eSCI_LCR1[LRES]) to abort the transmission, the LIN Wakeup Receive Flag in the LIN StatusRegister may be set (LWAKE=1). To avoid this, if the application has triggered LIN Protocol Engine Reset via the eSCI_LCR1[LRES], it should wait for the duration of a frame and clear the eSCI_IFSR2[LWAKE] flag before waiting for a wakeup.

NOTE

If the eSCI module is in LIN mode and is transmitting a LIN frame, and the application sets and subsequently clears the LIN reset bit (LRES) in the LIN Control register 1 (ESCI_LCR1), the next LIN frame transmission might incorrectly signal the occurrence of bit errors (ESCI_IFSR1[BERR]) and frame error (ESCI_IFSR1[FE]), and the transmitted frame might be incorrect.

31.4.6.7 LIN Protocol Engine Reset

The LIN protocol engine is reset when the LRES bit in the eSCI LIN Control Register 1 (eSCI_LCR1) is set to 1. In this case, the LIN protocol engine will no longer initiate new transmissions or receptions. However, ongoing byte transmission or reception is not halted.

In order to start the LIN Protocol Engine with idle transmitter and receiver processes, the LRES bit should be asserted for the duration of at least one bit.

31.4.7 Interrupts

This section describes the interrupt sources and interrupt request generation.

31.4.7.1 Interrupt Flags and Enables

All interrupt sources, interrupt flags, and interrupt enable bits are listed in Table 31-33. This table indicates the operational modes, where the interrupt flags can be set by the eSCI module.

Table 31-33. eSCI Interrupt Flags and Interrupt Enable Bits

Interrupt Source	Interrupt Flag	Interrupt Enable	Interrupt Enable Bit
Transmitter	SCI	eSCI_IFSR1[TDRE]	eSCI_CR1[TIE]
Transmitter	SCI, LIN	eSCI_IFSR1[TC]	eSCI_CR1[TCIE]
Receiver	SCI	eSCI_IFSR1[RDRF]	eSCI_CR1[RIE]
Receiver	SCI	eSCI_IFSR1[IDLE]	eSCI_CR1[ILIE]

Table 31-33. eSCI Interrupt Flags and Interrupt Enable Bits (continued)

Interrupt Source	Interrupt Flag	Interrupt Enable	Interrupt Enable Bit
Receiver	SCI	eSCI_IFSR1[OR]	eSCI_CR2[ORIE]
Receiver	SCI, LIN	eSCI_IFSR1[NF]	eSCI_CR2[NFIE]
Receiver	SCI, LIN	eSCI_IFSR1[FE]	eSCI_CR2[FEIE]
Receiver	SCI	eSCI_IFSR1[PF]	eSCI_CR2[PFIE]
Receiver	LIN	eSCI_IFSR1[BERR]	eSCI_CR2[BERRIE]
Receiver	LIN	eSCI_IFSR2[RXRDY]	eSCI_LCR1[RXIE]
Transmitter	LIN	eSCI_IFSR2[TXRDY]	eSCI_LCR1[TXIE]
Receiver	LIN	eSCI_IFSR2[LWAKE]	eSCI_LCR1[WUIE]
Receiver	LIN	eSCI_IFSR2[STO]	eSCI_LCR1[STIE]
Receiver	LIN	eSCI_IFSR2[PBERR]	eSCI_LCR1[PBIE]
Receiver	LIN	eSCI_IFSR2[CERR]	eSCI_LCR1[CIE]
Receiver	LIN	eSCI_IFSR2[CKERR]	eSCI_LCR1[CKIE]
Receiver	LIN	eSCI_IFSR2[FRC]	eSCI_LCR1[FCIE]
Receiver	LIN	eSCI_IFSR2[UREQ]	eSCI_LCR2[URIE]
Transmitter, Receiver	LIN	eSCI_IFSR2[OVFL]	eSCI_LCR2[OFIE]

31.4.7.2 Interrupt Request Generation

The eSCI module provides one hardware interrupt request signal to the systems interrupt controller. This interrupt request signal is asserted if and only if at least one of the interrupt flags and the corresponding interrupt enables are set to 1. Otherwise the interrupt line is deasserted.

31.5 Application Information

31.5.1 SCI Data Frames Separated by Preamble

To separate SCI data frame with preambles with minimum idle line time, use this sequence between messages:

1. Write to the eSCI SCI Data Register (eSCI_SDR).
   — This sets the internal iCMT bit, which requests the data transmission.
2. Wait until TDRE in the eSCI Interrupt Flag and Status Register 1 (eSCI_IFSR1) is set.
   — This indicates the start of transmission; the iCMT bit was cleared.
3. Clear and subsequently set the TE bit in the eSCI Control Register 1 (eSCI_CR1).
   — This set the internal iPRE bit, which requests the preamble transmission.
4. Write to the eSCI SCI Data Register (eSCI_SDR).
   — This sets the internal iCMT bit, which requests the data transmission.

The priority scheme of the transmitter (described in Table 31-25) ensures that the preamble is transmitted before the data frame.

Chapter 32
Inter-Integrated Circuit Bus Controller Module (I^2C)

32.1 Introduction

The inter-integrated circuit (I^2C™) bus is a two-wire bidirectional serial bus that provides a simple and efficient method of data exchange between devices. It minimizes the external connections to devices and does not require an external address decoder.

This bus is suitable for applications requiring occasional communication over a short distance between a number of devices. It also provides flexibility, allowing additional devices to be connected to the bus for further expansion and system development.

The interface is designed to operate as fast as 100 kbps with maximum bus loading and timing. The device is capable of operating at higher baud rates, as fast as a maximum of module clock/20, with reduced bus loading. The maximum communication length and the number of devices that can be connected are limited by a maximum bus capacitance of 400 pF.

The PXN20 provides four functionally identical I^2C modules, identified as I^2C_A through I^2C_D.

32.1.1 Block Diagram

A simplified block diagram of the I^2C illustrates the functionality and interdependence of major blocks (see Figure 32-1).

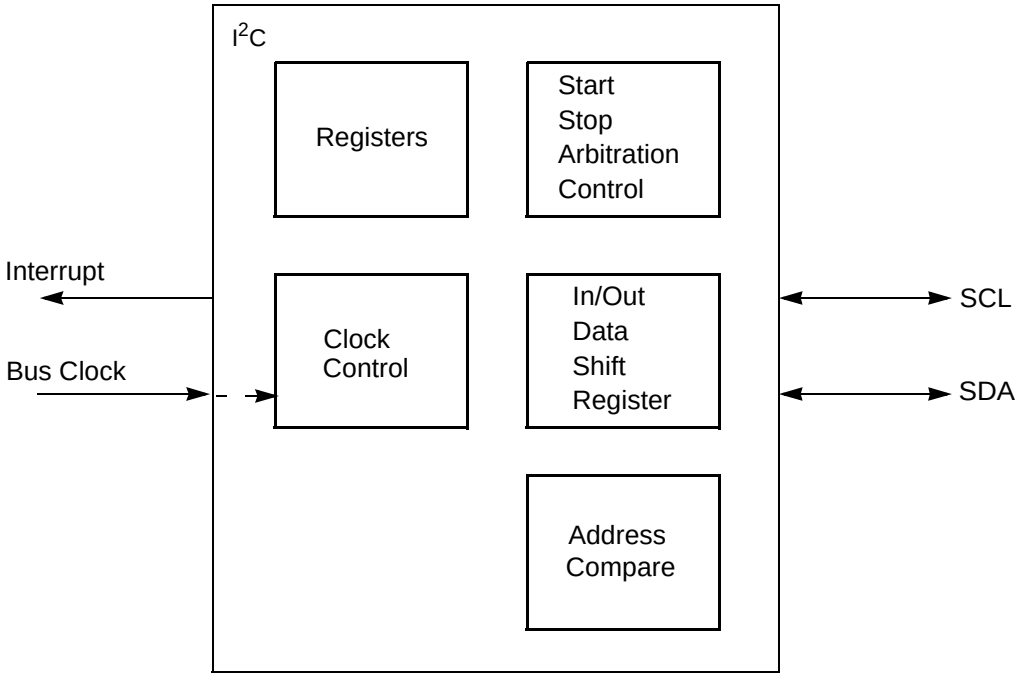


Figure 32-1. I^2C Block Diagram

32.1.2 DMA Interface

A simple DMA interface is implemented so that the I^2C can request data transfers with minimal support from the CPU. DMA mode is enabled by setting the DMAEN bit in the I^2C Bus Control Register (IBCR). DMA requests can be performed on all four I^2C channels.

The DMA interface is only valid when the I^2C module is configured for master mode and the DMA channel mux has selected the I^2C DMA request signals to be inputs to a DMA channel.

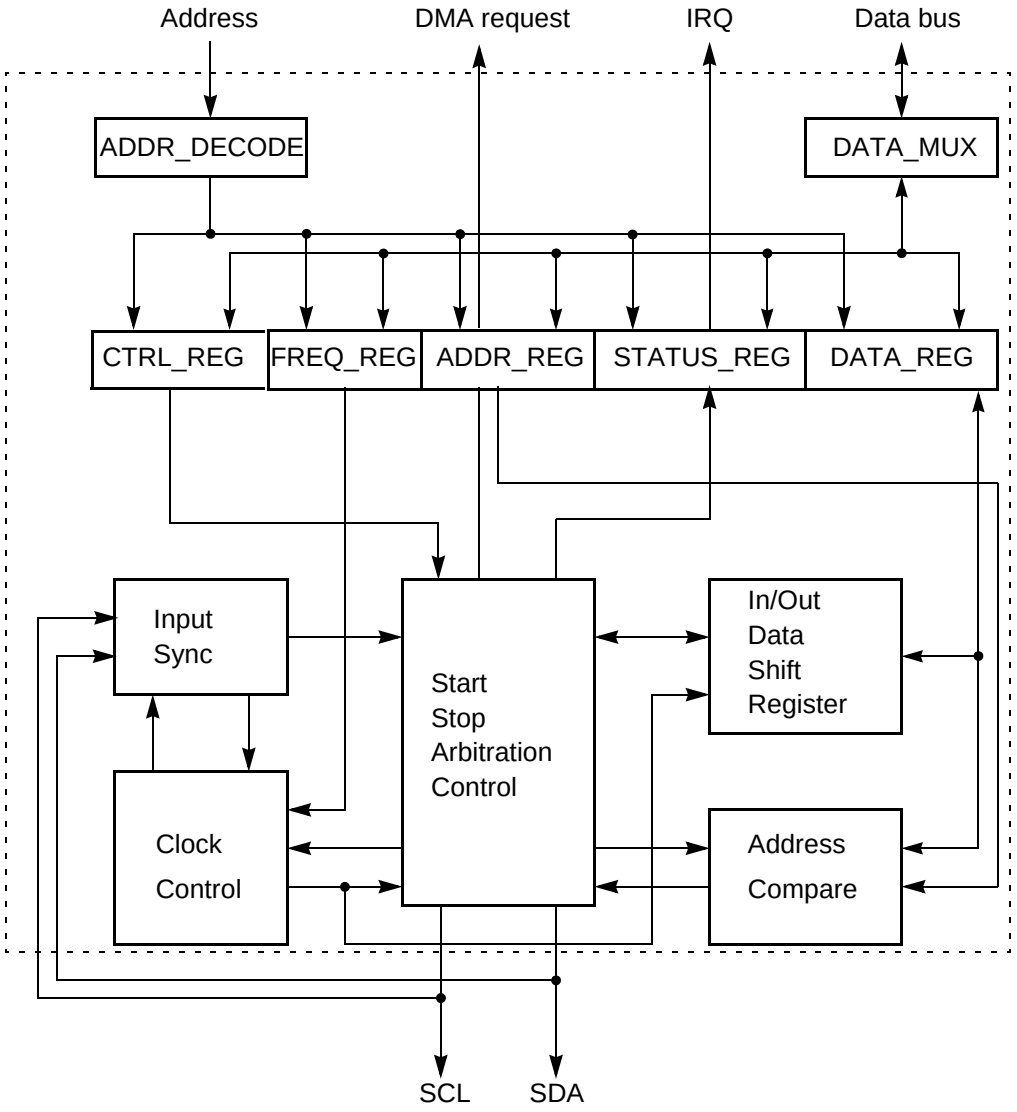


Figure 32-2. I^2C Module DMA Interface Block Diagram

32.1.3 Features

The I^2C has these major features:

- Compatible with I^2C bus standard
- Multi-master operation
- Software programmable for one of 256 serial clock frequencies
- Software selectable acknowledge bit
- Interrupt driven byte-by-byte data transfer
- Arbitration lost interrupt with automatic mode switching from master to slave
- Calling address identification interrupt
- Start and stop signal generation/detection

- Repeated start signal generation
- Acknowledge bit generation/detection
- Bus busy detection
- Basic DMA interface

Features currently not supported:

- No support for general call address
- Not compliant to ten-bit addressing

32.1.4 Modes of Operation

There are two operating modes of the I^2C module: run mode and halt mode. In run mode, I^2C_x = 0 in the SIU_HLT0 register and all functional parts of the I^2C module are running. In halt mode, I^2C_x = 1 in the SIU_HLT0 register and all clocks to the I^2C module are disabled.

32.2 External Signal Description

Refer to Chapter 3, Signal Description, for detailed signal descriptions.

32.3 Memory Map and Registers

This section provides a detailed description of all I^2C registers.

32.3.1 Module Memory Map

Table 32-1 shows the I^2C memory map. The address of each register is given as an offset to the I^2C base address. Registers are listed in address order, identified by complete name and mnemonic.

Table 32-1. I^2C Memory Map

Offset from I^2C_BASE I^2C_A = 0xFFF8_8000 I^2C_B = 0xFFF8_C000 I^2C_C = 0xC3F8_8000 I^2C_D = 0xC3F8_C000	Register	Access	Reset Value	Section/Page
0x0000	IBAD—I^2C bus address register	R/W	0x00	32.3.2.1/32-5
0x0001	IBFD—I^2C bus frequency divider register	R/W	0x00	32.3.2.2/32-5
0x0002	IBCR—I^2C bus control register	R/W	0x80	32.3.2.3/32-8
0x0003	IBSR—I^2C bus status register	R/W	0x80	32.3.2.4/32-9
0x0004	IBDR—I^2C bus data I/O register	R/W	0x00	32.3.2.5/32-10
0x0005	IBIC—I^2C bus interrupt configuration register	R/W	0x00	32.3.2.6/32-11
0x0006–0x3FFF	Reserved			

32.3.2 Register Descriptions

This section lists the I^2C registers in address order and describes the registers and their bit fields.

32.3.2.1 I^2C Bus Address Register (IBAD)

This register contains the address the I^2C bus responds to when addressed as a slave; it is not the address sent on the bus during the address transfer.

Offset: 0x00000 Access: User read/write

	0	1	2	3	4	5	6	7
R	AD							0
W								
Reset	0	0	0	0	0	0	0	0

Figure 32-3. I^2C Bus Address Register (IBAD)

Table 32-2. IBAD Field Descriptions

Field	Description
AD	Slave Address. Specific slave address to be used by the I^2C bus module. **Note:** The default mode of I^2C bus is slave mode for an address match on the bus.

32.3.2.2 I^2C Bus Frequency Divider Register (IBFD)

Offset: 0x0001 Access: User read/write

	0	1	2	3	4	5	6	7
R	MULT		ICR					
W								
Reset	0	0	0	0	0	0	0	0

Figure 32-4. I^2C Bus Frequency Divider Register (IBFD)

Table 32-3. IBFD Field Descriptions

Field	Description
MULT	I^2C Multiplier Factor. The MULT bits define the multiplier factor mul. This factor is used along with the SCL divider to generate the I^2C baud rate. The multiplier factor mul as defined by the MULT bits is provided below. 00 mul = 1 01 mul = 2 10 mul = 4 11 Reserved
ICR	I^2C Bus Clock Rate. The ICR bits are used to prescale the bus clock for bit rate selection. These bits and the MULT bits are used to determine the I^2C baud rate, the SDA hold time, the SCL Start hold time and the SCL Stop hold time. Table 32-4 provides the SCL divider and hold values for corresponding values of the ICR. The SCL divider multiplied by multiplier factor mul is used to generate I^2C baud rate. **I^2C baud rate = bus speed (Hz)/(mul * SCL divider)** ***Eqn. 32-1*** SDA hold time is the delay from the falling edge of SDA (I^2C data) to the changing of SDA (I^2C data). **SDA hold time = bus period (s) * mul * SDA hold value** ***Eqn. 32-2*** SCL Start hold time is the delay from the falling edge of SDA (I^2C data) while SCL is high (Start condition) to the falling edge of SCL (I^2C clock). **SCL Start hold time = bus period (s) * mul * SCL Start hold value** ***Eqn. 32-3*** SCL Stop hold time is the delay from the rising edge of SCL (I^2C clock) to the rising edge of SDA SDA (I^2C data) while SCL is high (Stop condition). **SCL Stop hold time = bus period (s) * mul * SCL Stop hold value** ***Eqn. 32-4***

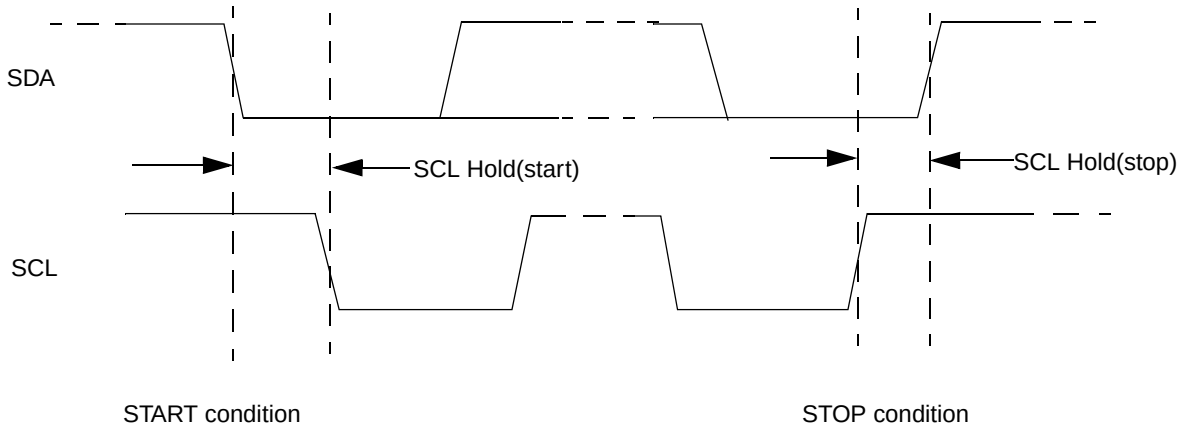


Figure 32-5. SCL Divider and SDA Hold

Table 32-4. I^2C Divider and Hold Values

ICR (hex)	SCL Divider	SDA Hold Value	SCL Hold (Start) Value	SDA Hold (Stop) Value
00	20	7	6	11
01	22	7	7	12
02	24	8	8	13
03	26	8	9	14
04	28	9	10	15
05	30	9	11	16
06	34	10	13	18
07	40	10	16	21
08	28	7	10	15
09	32	7	12	17
0A	36	9	14	19
0B	40	9	16	21
0C	44	11	18	23
0D	48	11	20	25
0E	56	13	24	29
0F	68	13	30	35
10	48	9	18	25
11	56	9	22	29
12	64	13	26	33
13	72	13	30	37
14	80	17	34	41
15	88	17	38	45
16	104	21	46	53
17	128	21	58	65
18	80	9	38	41
19	96	9	46	49
1A	112	17	54	57
1B	128	17	62	65
1C	144	25	70	73
1D	160	25	78	81
1E	192	33	94	97
1F	240	33	118	121

ICR (hex)	SCL Divider	SDA Hold Value	SCL Hold (Start) Value	SCL Hold (Stop) Value
20	160	17	78	81
21	192	17	94	97
22	224	33	110	113
23	256	33	126	129
24	288	49	142	145
25	320	49	158	161
26	384	65	190	193
27	480	65	238	241
28	320	33	158	161
29	384	33	190	193
2A	448	65	222	225
2B	512	65	254	257
2C	576	97	286	289
2D	640	97	318	321
2E	768	129	382	385
2F	960	129	478	481
30	640	65	318	321
31	768	65	382	385
32	896	129	446	449
33	1024	129	510	513
34	1152	193	574	577
35	1280	193	638	641
36	1536	257	766	769
37	1920	257	958	961
38	1280	129	638	641
39	1536	129	766	769
3A	1792	257	894	897
3B	2048	257	1022	1025
3C	2304	385	1150	1153
3D	2560	385	1278	1281
3E	3072	513	1534	1537
3F	3840	513	1918	1921

32.3.2.3 I^2C Bus Control Register (IBCR)

Offset: 0x0002 Access: User read/write

	0	1	2	3	4	5	6	7
R	MDIS	IBIE	MS	TX	NOACK	0	DMAEN	0
W						RSTA		
Reset	1	0	0	0	0	0	0	0

Figure 32-6. I^2C Bus Control Register (IBCR)

Table 32-5. IBCR Field Descriptions

Field	Description
MDIS	Module Disable. This bit controls the software reset of the entire I^2C bus module. 0 The I^2C bus module is enabled. This bit must be cleared before any other IBCR bits have any effect. 1 The module is reset and disabled. This is the power-on reset situation. When high, the interface is held in reset, but registers can be accessed. **Note:** If the I^2C bus module is enabled in the middle of a byte transfer, the interface behaves as follows: slave mode ignores the current transfer on the bus and starts operating when a subsequent start condition is detected. Master mode is not aware that the bus is busy. Therefore, if a start cycle is initiated, the current bus cycle may become corrupt. This ultimately results in the current bus master or the I^2C bus module losing arbitration, after which, bus operation returns to normal.
IBIE	I-Bus Interrupt Enable. 0 Interrupts from the I^2C bus module are disabled. This does not clear any currently pending interrupt condition. 1 Interrupts from the I^2C bus module are enabled. An I^2C bus interrupt occurs provided the IBIF bit in the status register is also set.
MS	Master/Slave Mode Select. This bit is cleared on reset. When this bit is changed from 0 to 1, a START signal is generated on the bus and the master mode is selected. When this bit is changed from 1 to 0, a STOP signal is generated and the operation mode changes from master to slave. A STOP signal should be generated if only the IBIF flag is set. MS is cleared without generating a STOP signal when the master loses arbitration. 0 Slave mode. 1 Master mode.
TX	Transmit/Receive Mode Select. This bit selects the direction of master and slave transfers. When addressed as a slave this bit must be set by software according to the SRW bit in the status register. In master mode this bit must be set according to the type of transfer required. Therefore, for address cycles, this bit is always high. 0 Receive. 1 Transmit.
NOACK	Data Acknowledge Disable. This bit specifies the value driven onto SDA during data acknowledge cycles for both master and slave receivers. The I^2C module always acknowledges address matches, provided it is enabled, regardless of the value of NOACK. Values written to this bit are used only when the I^2C Bus is a receiver, not a transmitter. 0 An acknowledge signal is sent out to the bus at the 9th clock bit after receiving one byte of data. 1 No acknowledge signal response is sent (i.e., acknowledge bit = 1).

Table 32-5. IBCR Field Descriptions (continued)

Field	Description
RSTA	Repeat Start. Writing a 1 to this bit generates a repeated START condition on the bus, provided it is the current bus master. This bit is always read as a low. Attempting a repeated start at the wrong time, if the bus is owned by another master, results in loss of arbitration. 0 No effect. 1 Generate repeat start cycle.
DMAEN	DMA enable. When this bit is set, the DMA TX and RX lines are asserted when the I^2C module requires data to be read or written to the data register. No transfer done interrupts are generated when this bit is set; however, an interrupt is generated if loss of arbitration or addressed as slave conditions occur. The DMA mode is valid only when the I^2C module is configured as a master and the DMA transfer still requires CPU intervention at the start and the end of each frame of data. See the DMA application information section for more details. 0 Disable the DMA TX/RX request signals. 1 Enable the DMA TX/RX request signals.

32.3.2.4 I^2C Bus Status Register (IBSR)

Offset: 0x0003 Access: User read/write

	0	1	2	3	4	5	6	7
R	TCF	IAAS	IBB	IBAL	0	SRW	IBIF	RXAK
W							w1c	
Reset	1	0	0	0	0	0	0	0

Figure 32-7. I^2C Bus Status Register (IBSR)

Table 32-6. IBSR Field Descriptions

Field	Description
TCF	Transfer Complete. While one byte of data is transferred, this bit is cleared. It is set by the falling edge of the ninth clock of a byte transfer. This bit is valid only during or immediately following a transfer to the I^2C module or from the I^2C module. 0 Transfer in progress. 1 Transfer complete.
IAAS	Addressed as a Slave. When its own specific address (I-bus address register) is matched with the calling address, this bit is set. The CPU is interrupted provided the IBIE is set. Then the CPU must check the SRW bit and set its Tx/$\overline{Rx}$ mode accordingly. Writing to the I-bus control register clears this bit. 0 Not addressed. 1 Addressed as a slave.
IBB	Bus Busy. This bit indicates the status of the bus. When a START signal is detected, the IBB is set. If a STOP signal is detected, IBB is cleared and the bus enters idle state. 0 Bus is idle. 1 Bus is busy.
IBAL	Arbitration Lost. The arbitration lost bit (IBAL) is set by hardware when the arbitration procedure is lost. Arbitration is lost in the following circumstances: • SDA is sampled low when the master drives a high during an address or data transmit cycle. • SDA is sampled low when the master drives a high during the acknowledge bit of a data receive cycle. • A start cycle is attempted when the bus is busy. • A repeated start cycle is requested in slave mode. • A stop condition is detected when the master did not request it. This bit must be cleared by software, by writing a one to it. A write of zero has no effect.

Table 32-6. IBSR Field Descriptions (continued)

Field	Description
bit 4	Reserved for future use. A read returns 0; must be written as 0.
SRW	Slave Read/Write. When IAAS is set, this bit indicates the value of the R/W command bit of the calling address sent from the master. This bit is valid only when the I-bus is in slave mode, a complete address transfer has occurred with an address match and no other transfers have been initiated. By programming this bit, the CPU can select slave transmit/receive mode according to the command of the master. 0 Slave receive, master writing to slave. 1 Slave transmit, master reading from slave.
IBIF	I-Bus Interrupt Flag. The IBIF bit is set when one of the following conditions occurs: • Arbitration lost (IBAL bit set) • Byte transfer complete (TCF bit set and DMAEN bit not set) • Addressed as slave (IAAS bit set) • NoAck from slave (MS and TX bits set) • I^2C bus going idle (IBB high-low transition and enabled by BIIE) A processor interrupt request is generated if the IBIE bit is set. This bit must be cleared by software, by writing a 1 to it. A write of 0 has no effect on this bit. In DMA mode (DMAEN set), a byte transfer complete condition does not trigger the setting of IBIF. All other conditions apply.
RXAK	Received Acknowledge. This is the value of SDA during the acknowledge bit of a bus cycle. If the received acknowledge bit (RXAK) is low, it indicates an acknowledge signal has been received after the completion of 8 bits data transmission on the bus. If RXAK is high, it means no acknowledge signal is detected at the 9th clock. 0 Acknowledge received. 1 No acknowledge received.

32.3.2.5 I^2C Bus Data I/O Register (IBDR)

Offset: 0x0004 Access: User read/write

	0	1	2	3	4	5	6	7
R W	Data							
Reset	0	0	0	0	0	0	0	0

Figure 32-8. I^2C Bus Data I/O Register (IBDR)

In master transmit mode, when data is written to IBDR, a data transfer is initiated. The most significant bit is sent first. In master receive mode, reading this register initiates next byte data receiving. In slave mode, the same functions are available after an address match has occurred. The TX bit in the IBCR must correctly reflect the desired direction of transfer in master and slave modes for the transmission to begin. For instance, if the I^2C is configured for master transmit but a master receive is desired, then reading the IBDR does not initiate the receive.

Reading the IBDR returns the last byte received while the I^2C is configured in either master receive or slave receive modes. The IBDR does not reflect every byte that is transmitted on the I^2C bus, nor can software verify that a byte has been written to the IBDR correctly by reading it back.

In master-transmit mode, the first byte of data written to IBDR following assertion of MS is used for the address transfer and should comprise the calling address (in position D0–D6) concatenated with the required R/$\overline{W}$ bit (in position D7).

32.3.2.6 I^2C Bus Interrupt Configuration Register (IBIC)

Offset: 0x0005 Access: User read/write

	0	1	2	3	4	5	6	7
R	BIIE	0	0	0	0	0	0	0
W								
Reset	0	0	0	0	0	0	0	0

Figure 32-9. I^2C Bus Interrupt Configuration Register (IBIC)

Table 32-7. IBIC Field Descriptions

Field	Description
BIIE	Bus Idle Interrupt Enable Bit. This configuration bit can be used to enable the generation of an interrupt after the I^2C bus becomes idle. After this bit is set, an IBB high-low transition sets the IBIF bit. This feature can be used to signal to the CPU the completion of a STOP on the I^2C bus. 0 Bus idle interrupts disabled. 1 Bus idle interrupts enabled.
bits 1–7	Reserved for future use. A read returns 0; must be written as 0.

32.4 Functional Description

32.4.1 I-Bus Protocol

The I^2C bus system uses a serial data line (SDA) and a serial clock line (SCL) for data transfer. All devices connected to it must have open-drain or open-collector outputs. A logical AND function is exercised on both lines with external pullup resistors. The value of these resistors is system dependent.

Normally, a standard communication is composed of four parts: START signal, slave address transmission, data transfer, and STOP signal. They are described briefly in the following sections and illustrated in Figure 32-10.

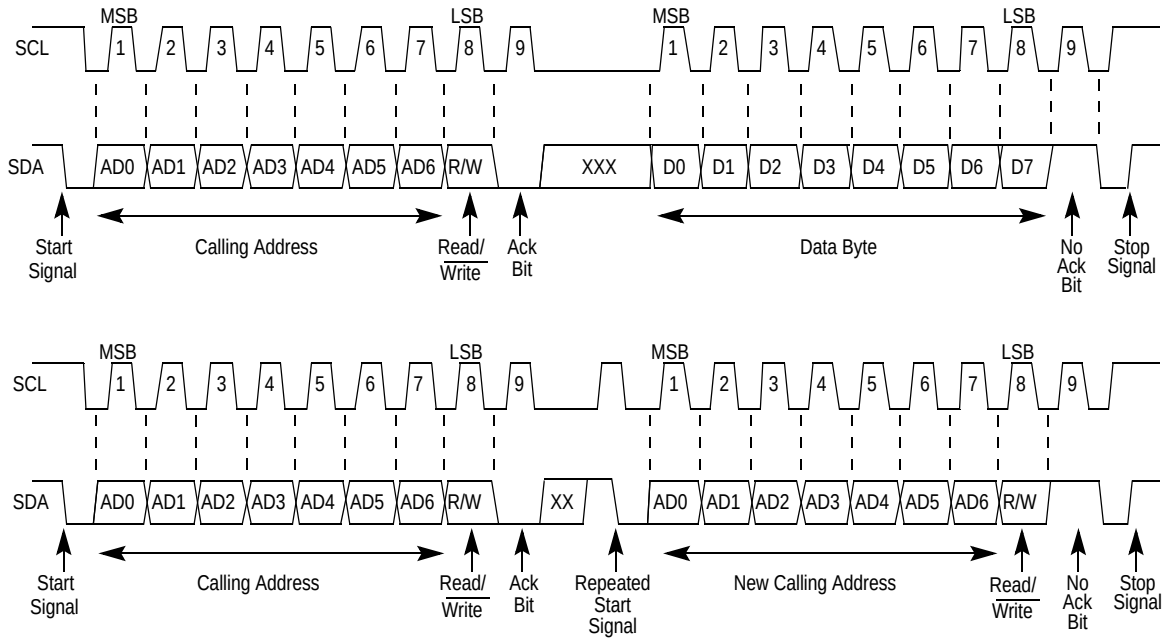


Figure 32-10. I^2C Bus Transmission Signals

32.4.1.1 START Signal

When the bus is free, i.e.,no master device is engaging the bus (both SCL and SDA lines are at logical high), a master may initiate communication by sending a START signal. As shown in Figure 32-10, a START signal is a high-to-low transition of SDA while SCL is high. This signal denotes the beginning of a new data transfer (each data transfer may contain several bytes of data) and brings all slaves out of their idle states.

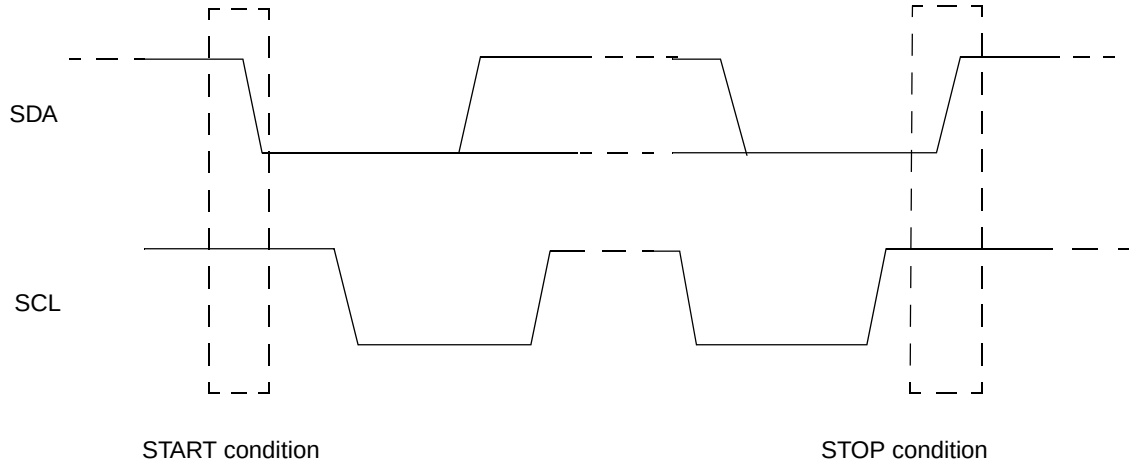


Figure 32-11. Start and Stop conditions

32.4.1.2 Slave Address Transmission

The first byte of data transfer immediately after the START signal is the slave address transmitted by the master. This is a seven-bit calling address followed by a R/W bit. The R/W bit tells the slave the desired direction of data transfer.

1 = Read transfer—the slave transmits data to the master

0 = Write transfer—the master transmits data to the slave

Only the slave with a calling address that matches the one transmitted by the master responds by sending back an acknowledge bit. This is done by pulling the SDA low at the ninth clock (see Figure 32-10).

No two slaves in the system may have the same address. If the I^2C bus is master, it must not transmit an address that is equal to its own slave address. The I^2C bus cannot be master and slave at the same time. However, if arbitration is lost during an address cycle, the I^2C bus reverts to slave mode and operates correctly, even if it is being addressed by another master.

32.4.1.3 Data Transfer

After successful slave addressing is achieved, the data transfer can proceed byte-by-byte in a direction specified by the R/W bit sent by the calling master.

All transfers that come after an address cycle are referred to as data transfers, even if they carry sub-address information for the slave device.

Each data byte is 8 bits long. Data may be changed only while SCL is low and must be held stable while SCL is high (see Figure 32-10). There is one clock pulse on SCL for each data bit, the MSB being transferred first. Each data byte must be followed by an acknowledge bit, which is signalled from the receiving device by pulling the SDA low at the ninth clock. Therefore, one complete data byte transfer needs nine clock pulses.

If the slave receiver does not acknowledge the master, the SDA line must be left high by the slave. The master can then generate a stop signal to abort the data transfer or a start signal (repeated start) to commence a new calling.

If the master receiver does not acknowledge the slave transmitter after a byte transmission, it means end of data to the slave, so the slave releases the SDA line for the master to generate a STOP or START signal.

32.4.1.4 STOP Signal

The master can terminate the communication by generating a STOP signal to free the bus. However, the master may generate a START signal followed by a calling command without generating a STOP signal first. This is called repeated START. A STOP signal is defined as a low-to-high transition of SDA while SCL is at logical 1 (see Figure 32-10).

The master can generate a STOP even if the slave has generated an acknowledge, at which point the slave must release the bus.

32.4.1.5 Repeated START Signal

As shown in Figure 32-10, a repeated START signal is a START signal generated without first generating a STOP signal to terminate the communication. This is used by the master to communicate with another slave or with the same slave in different mode (transmit/receive mode) without releasing the bus.

32.4.1.6 Arbitration Procedure

The Inter-IC bus is a true multi-master bus that allows more than one master to be connected on it. If two or more masters try to control the bus simultaneously, a clock synchronization procedure determines the bus clock, for which the low period is equal to the longest clock low period and the high is equal to the shortest among the masters. The relative priority of the contending masters is determined by a data arbitration procedure. A bus master loses arbitration if it transmits logic “1” while another master transmits logic 0. The losing masters immediately switch to slave receive mode and stop driving the SDA output. In this case, the transition from master to slave mode does not generate a STOP condition. Meanwhile, a status bit is set by hardware to indicate loss of arbitration.

32.4.1.7 Clock Synchronization

Because wire-AND logic is performed on the SCL line, a high-to-low transition on the SCL line affects all the devices connected on the bus. The devices start counting their low period and after a device's clock has gone low, it holds the SCL line low until the clock high state is reached. However, the change of low to high in this device clock may not change the state of the SCL line if another device clock remains within its low period. Therefore, synchronized clock SCL is held low by the device with the longest low period. Devices with shorter low periods enter a high wait state during this time (see Figure 32-12). When all engaged devices have counted off their low period, the synchronized clock SCL line is released and pulled high. There is then no difference between the device clocks and the state of the SCL line and all the devices start counting their high periods. The first device to complete its high period pulls the SCL line low again.

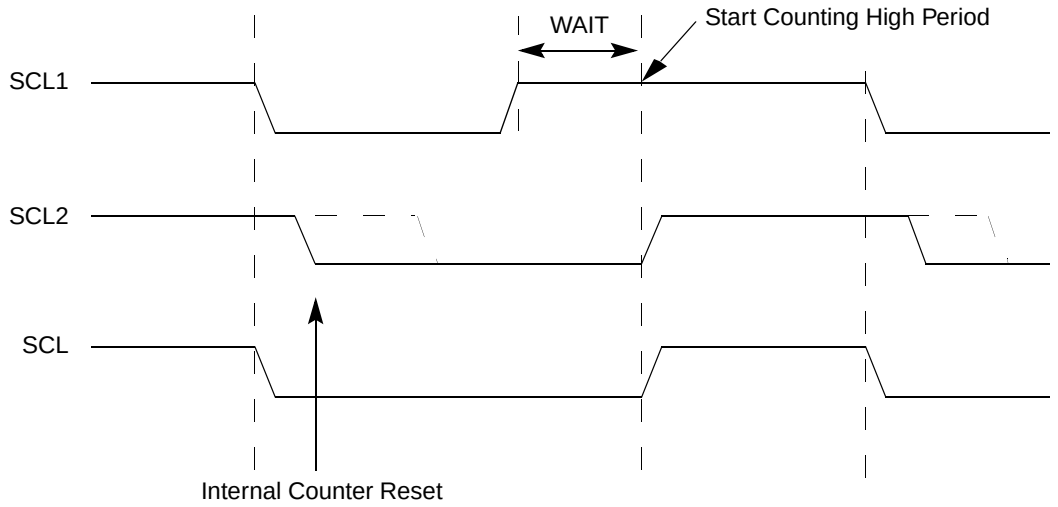


Figure 32-12. I^2C Bus Clock Synchronization

32.4.1.8 Handshaking

The clock synchronization mechanism can be used as a handshake in data transfer. Slave devices may hold the SCL low after completion of one byte transfer (9 bits). In such cases, it halts the bus clock and forces the master clock into wait state until the slave releases the SCL line.

32.4.1.9 Clock Stretching

The clock synchronization mechanism can be used by slaves to slow the bit rate of a transfer. After the master has driven SCL low, the slave can drive SCL low for the required period and then release it. If the slave SCL low period is greater than the master SCL low period then the resulting SCL bus signal low period is stretched.

32.4.2 Interrupts

32.4.2.1 General

The I^2C uses one interrupt vector only.

Table 32-8. Interrupt Summary

Interrupt	Offset	Vector	Priority	Source	Description
I^2C Interrupt	—	—	—	IBAL, TCF, IAAS, IBB bits in IBSR register	When any IBAL, TCF, or IAAS bits are set an interrupt may be caused based on arbitration lost, transfer complete or address detect conditions. If enabled by BIIE, the deassertion of IBB can also cause an interrupt, indicating that the bus is idle.

32.4.2.2 Interrupt Description

There are five types of internal interrupts in the I^2C. The interrupt service routine can determine the interrupt type by reading the status register.

I^2C Interrupt can be generated on:

- Arbitration lost condition (IBAL bit set)
- Byte transfer condition (TCF bit set and DMAEN bit not set)
- Address detect condition (IAAS bit set)
- No acknowledge from slave received when expected
- Bus going idle (IBB bit not set)

The I^2C interrupt is enabled by the IBIE bit in the I^2C control register. It must be cleared by writing 1 to the IBIF bit in the interrupt service routine. The bus going idle interrupt needs to be additionally enabled by the BIIE bit in the IBIC register.

32.5 Initialization/Application Information

32.5.1 I^2C Programming Examples

32.5.1.1 Initialization Sequence

Reset puts the I^2C bus control register in its default state. Before the interface can be used to transfer serial data, an initialization procedure must be carried out, as follows:

1. Update the frequency divider register (IBFD) and select the required division ratio to obtain SCL frequency from system clock.
2. Update the I^2C bus address register (IBAD) to define its slave address.
3. Clear the MDIS bit of the I^2C bus control register (IBCR) to enable the I^2C interface system.
4. Modify the bits of the IBCR to select master/slave mode, transmit/receive mode and interrupt enable or not. Optionally modify the bits of the I^2C bus interrupt configuration register (IBIC) to further refine the interrupt behavior.
5. Configure the SDA and SCL pads. (The SIU Pad Configuration registers must be configured to select the appropriate I^2C function. Also, the open drain feature of the pad must be enabled by setting the ODE bit in the appropriate SIU pad configuration register.)

32.5.1.2 Generation of START

After completion of the initialization procedure, serial data can be transmitted by selecting the master transmitter mode. If the device is connected to a multi-master bus system, the state of the I^2C bus busy bit (IBB) must be tested to check if the serial bus is free.

If the bus is free (IBB = 0), the start condition and the first byte (the slave address) can be sent. The data written to the data register comprises the slave calling address and the LSB, which is set to indicate the direction of transfer required from the slave.

The bus free time (i.e., the time between a STOP condition and the following START condition) is built into the hardware that generates the START cycle. Depending on the relative frequencies of the system clock and the SCL period, it may be necessary to wait until the I^2C is busy after writing the calling address to the IBDR before proceeding with the following instructions. This is illustrated in the following example.

An example of the sequence of events which generates the START signal and transmits the first byte of data (slave address) is shown below:

```
while (bit 2, IBSR ==1)     // wait in loop for IBB flag to clear
bit3 and bit 2, IBCR = 1    // set transmit and master mode, i.e. generate start condition
IBDR = calling_address      // send the calling address to the data register
while (bit 2, IBSR ==0)     // wait in loop for IBB flag to be set
```

32.5.1.3 Post-Transfer Software Response

Transmission or reception of a byte sets the data transferring bit (TCF) to 1, which indicates one byte communication is finished. The I^2C Bus interrupt bit (IBIF) is set also; an interrupt is generated if the

interrupt function is enabled during initialization by setting the IBIE bit. The IBIF (interrupt flag) can be cleared by writing 1 (in the interrupt service routine, if interrupts are used).

The TCF bit is cleared to indicate data transfer in progress by reading the IBDR data register in receive mode or writing the IBDR in transmit mode. The TCF bit must not be used as a data transfer complete flag because the flag timing depends on a number of factors including the I^2C bus frequency. This bit may not conclusively provide an indication of a transfer complete situation. Transfer complete situations must be detected using the IBIF flag

Software may service the I^2C I/O in the main program by monitoring the IBIF bit if the interrupt function is disabled. Polling should monitor the IBIF bit rather than the TCF bit because their operation is different when arbitration is lost.

When an interrupt occurs at the end of the address cycle, the master is always in transmit mode, i.e., the address is transmitted. If master receive mode is required, indicated by R/W bit in IBDR, then the TX bit should be toggled at this stage.

During slave mode address cycles (IAAS = 1) the SRW bit in the status register is read to determine the direction of the subsequent transfer and the TX bit is programmed accordingly. For slave mode data cycles (IAAS = 0) the SRW bit is not valid. The TX bit in the control register should be read to determine the direction of the current transfer.

The following is an example software sequence for master transmitter in the interrupt routine.

```
clear bit 6, IBSR           // Clear the IBIF flag
if (bit 2, IBCR ==0)
    slave_mode()            // run slave mode routine
if (bit 3, IBCR ==0))
    receive_mode()          // run receive_mode routine
if (bit 7, IBSR == 1)       // if NO ACK
     end();                 // end transmission
else
IBDR = data_to_transmit     // transmit next byte of data
```

32.5.1.4 Generation of STOP

A data transfer ends with a STOP signal generated by the master device. A master transmitter can simply generate a STOP signal after all the data has been transmitted. The following example shows how a stop condition is generated by a master transmitter.

```
if (tx_count == 0) or       // check to see if all data bytes have been transmitted
    (bit 7, IBSR == 1) {    // or if no ACK generated
         clear bit 2, IBCR  // generate stop condition
        }
else {
IBDR = data_to_transmit     // write byte of data to DATA register
          tx_count --       // decrement counter
      }                     // return from interrupt
```

If a master receiver wants to terminate a data transfer, it must inform the slave transmitter by not acknowledging the last byte of data. This can be done by setting the transmit acknowledge bit (TXAK) before reading the second last byte of data. Before reading the last byte of data, a STOP signal must first be generated. The following example shows how a STOP signal is generated by a master receiver.

```
rx_count --                 // decrease the rx counter
if (rx_count ==1)           // 2nd last byte to be read ?
   bit 4, IBCR = 1          // disable ACK
if (rx_count == 0)          // last byte to be read ?
  bit 6, IBCR = 0           // generate stop signal
else
data_received = IBDR        // read RX data and store
```

32.5.1.5 Generation of Repeated START

At the end of data transfer, if the master wants to remain communicating on the bus, it can generate another START signal followed by another slave address without first generating a STOP signal. A program example is as shown.

```
bit 5, IBCR = 1             // generate another start ( restart)
IBDR == calling_address     // transmit the calling address
```

32.5.1.6 Slave Mode

In the slave interrupt service routine, the module addressed as slave bit (IAAS) must be tested to check if a calling of its own address has been received. If IAAS is set, software sets the transmit/receive mode select bit (TX bit of IBCR) according to the R/W command bit (SRW). Writing to the IBCR clears IAAS automatically. IAAS is read as set when it is from the interrupt at the end of the address cycle where an address match occurred. Interrupts resulting from subsequent data transfers have IAAS cleared. A data transfer may be initiated by writing information to IBDR for slave transmits or dummy reading from IBDR in slave receive mode. The slave drives SCL low in-between byte transfers SCL is released when the IBDR is accessed in the required mode.

In slave transmitter routine, the received acknowledge bit (RXAK) must be tested before transmitting the next byte of data. Setting RXAK means an end of data signal from the master receiver, after which it must be switched from transmitter mode to receiver mode by software. A dummy read then releases the SCL line so the master can generate a STOP signal.

32.5.1.7 Arbitration Lost

If several masters try to engage the bus simultaneously, one master wins and the others lose arbitration. The devices that lost arbitration are immediately switched to slave receive mode by the hardware. Their data output to the SDA line is stopped, but SCL remains generated until the end of the byte during which arbitration was lost. An interrupt occurs at the falling edge of the ninth clock of this transfer with IBAL = 1 and MS = 0. If one master attempts to start transmission while the bus is being engaged by another master, the hardware inhibits the transmission, switches the MS bit from 1 to 0 without generating a STOP condition, generates an interrupt to CPU, and sets the IBAL to indicate that the attempt to engage the bus is failed. When considering these cases, the slave service routine should test the IBAL first and the software should clear the IBAL bit if it is set.

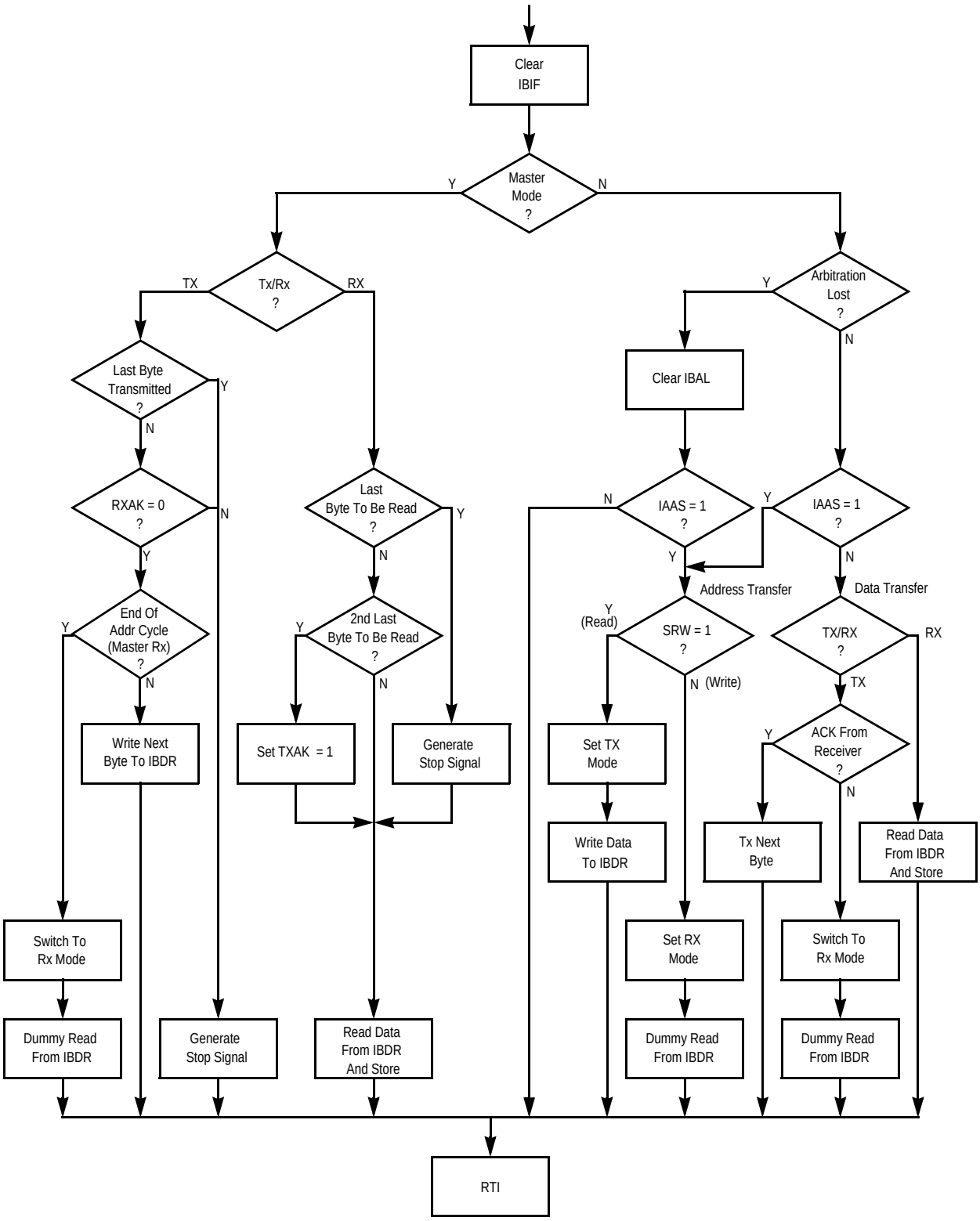


Figure 32-13. Flowchart of Typical I^2C Interrupt Routine

32.5.2 DMA Application Information

The DMA interface on the I^2C is not completely autonomous and requires intervention from the CPU to start and to terminate the frame transfer. DMA mode is valid for master-transmit and master-receive modes only. Software must ensure that the DMA enable bit in the control register is not set when the I^2C module is configured in master mode.

The DMA controller must transfer only one byte of data per Tx/Rx request. This is because there is no FIFO on the I^2C block.

The CPU should also keep the I^2C interrupt enabled during a DMA transfer to detect the arbitration lost condition and take action to recover from this situation. The DMAEN bit in the IBCR register works as a disable for the transfer complete interrupt. This means that during normal transfers (no errors) there always is either an interrupt or a request to the DMA controller, depending on the setting of the DMAEN bit. All error conditions trigger an interrupt and require CPU intervention. The address match condition does not occur in DMA mode as the I^2C should never be configured for slave operation.

The following sections detail how to set up a DMA transfer and what intervention is required from the CPU. It is assumed that the system DMA controller is capable of generating an interrupt after a certain number of DMA transfers have taken place.

32.5.2.1 DMA Mode, Master Transmit

Figure 32-14 details exactly the operation for using a DMA controller to transmit *n* data bytes to a slave. The first byte (the slave calling address) is always transmitted by the CPU. All subsequent data bytes (apart from the last data byte) can be transferred by the DMA controller. The last data byte must be transferred by the CPU.

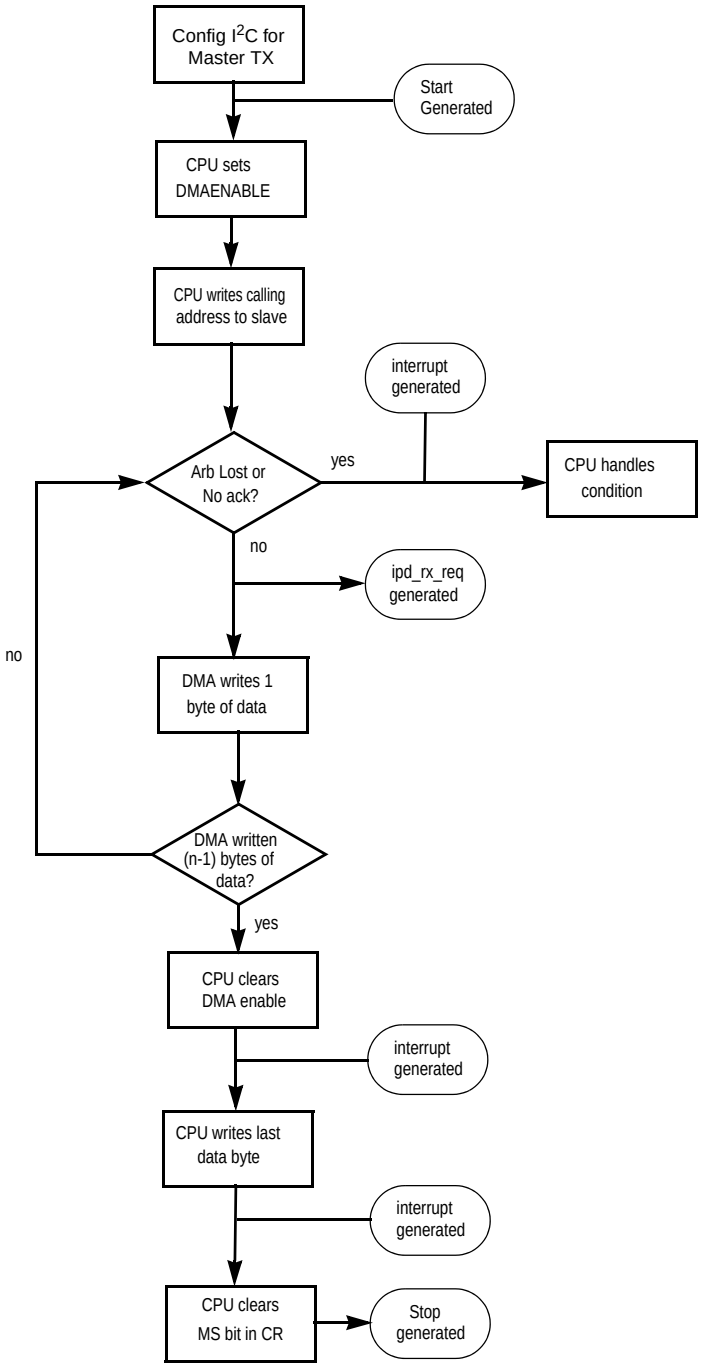


Figure 32-14. Flowchart of DMA Mode Master Transmit

32.5.2.2 DMA Mode, Master RX

Figure 32-15 details the exact operation for using a DMA controller to receive *n* data bytes from a slave. The first byte (the slave calling address) is always transmitted by the CPU. All subsequent data bytes (apart from the two last data bytes) can be read by the DMA controller. The last two data bytes must be transferred by the CPU.

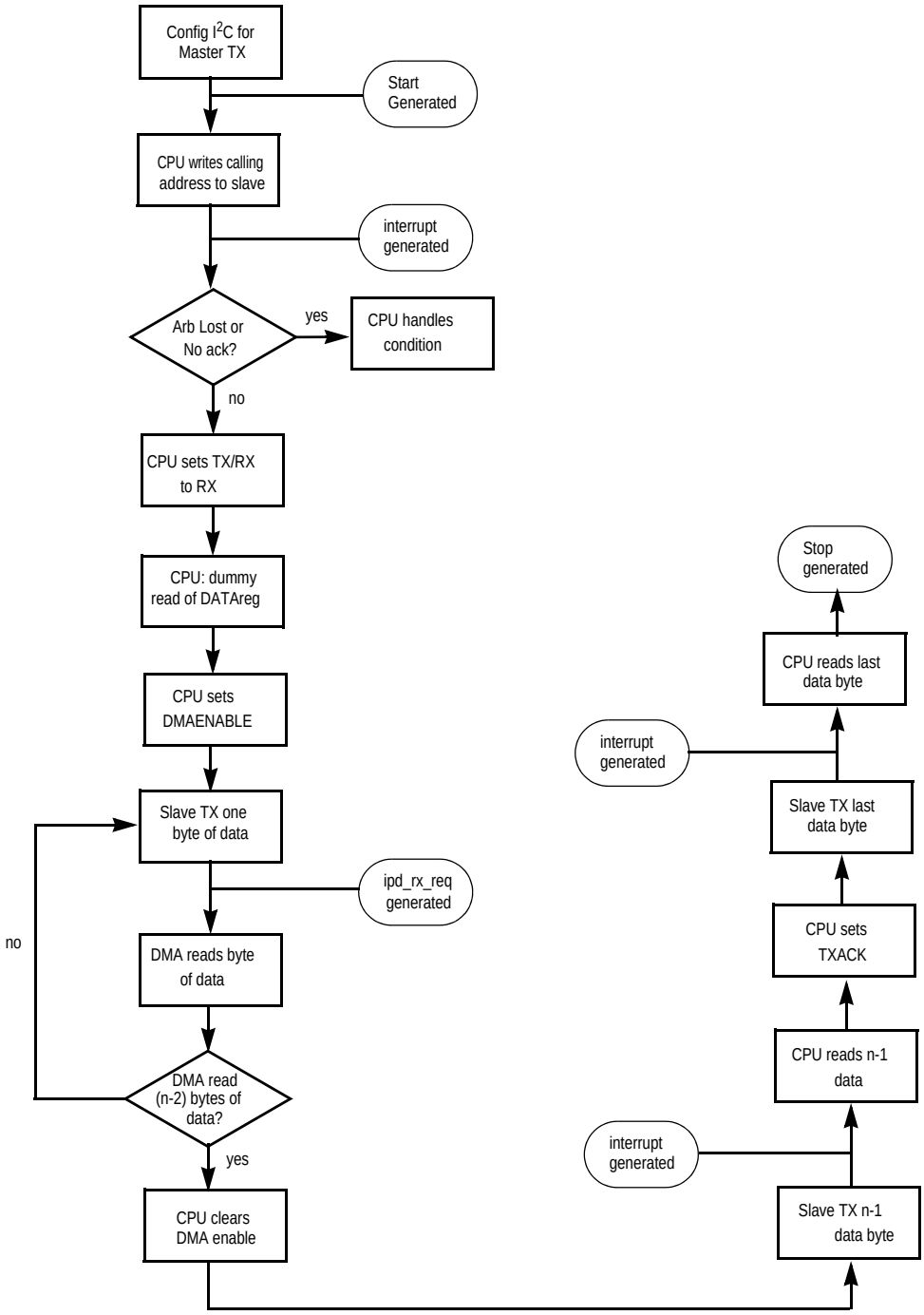


Figure 32-15. Flowchart of DMA Mode Master Receive

32.5.2.3 Exiting DMA Mode, System Requirement Considerations

As described above, the final transfers of both Tx and Rx transfers need to be managed via interrupt by the CPU. To change from DMA to interrupt driven transfers in the I^2C module, disable the DMAEN bit in the IBCR register. The trigger to exit the DMA mode is that the programmed DMA transfer control descriptor (TCD) has completed all its transfers to/from the I^2C module.

After the last DMA write (TX mode) to the I^2C the module immediately starts the next I^2C-bus transfer. The same is true for RX mode. After the DMA read from the IBDR register the module initiates the next I^2C-bus transfer. This results in two possible scenarios in the DMA mode exiting scheme.

1. Fast reaction

   The DMAEN bit is cleared before the next I^2C-bus transfer completes. In this case, the module raises an interrupt request to the CPU which can be serviced normally.

2. Slow reaction

   The DMAEN bit is cleared after the next I^2C-bus transfer has already completed. In this case, the module does not raise an interrupt request to the CPU. Instead, the TCF bit can be read to determine that the transfer completed and the module is ready for further transfer.

32.5.2.3.1 Fast vs. Slow Reaction

The reaction time T_R for the system to disable DMAEN after the last DMA controller access to the I^2C is the time required for one byte transfer over the I^2C. In a fast reaction the disabling has to occur before the ninth bit of the data transfer, which is the ACK bit. So the time available is eight times the SCL period.

$$T_R = 8 \times T_{SCL} \qquad \textit{Eqn. 32-5}$$

In fast mode, with 400 kbit/s, T_{SCL} is 2.5 µs, so T_R is 20 µs.

Depending on the system and DMA controller there are different possibilities for the deassertion of DMAEN. Three options are:

1. CPU intervention via interrupt

   The DMA controller is programmed to signal an interrupt to the CPU which is then responsible for the deassertion of DMAEN. This scheme is supported by most systems but can result in a slow reaction time if higher priority interrupts interfere. Therefore, the interrupt handling routine can become complicated as it has to check which of the two scenarios happened (check TCF bit) and act accordingly. In case of slow reaction you can force an interrupt for the I^2C in the interrupt controller to have the further transfer handled by the normal I^2C interrupt routine. The use of nested interrupts can cause problems in this scenario, if the DMA interrupt stalls between the deassertion and the DMAEN bit and the checking of the TCF bit.

2. DMA channel linking (if supported)

   The transfer control descriptor in the DMA controller that performs the data transfer is linked to another channel that does a write to the I^2C IBCR register to disable the DMAEN bit. This is probably the fastest system solution, but it uses two DMA channels. On the system level, no higher priority DMA requests must occur between the two linked TCDs because those can result in slow reaction.

3. DMA scatter/gather process (if supported)

   The transfer control descriptor in the DMA controller that performs the data transfer has the scatter-gather feature activated. This feature initiates a reload of another TCD from system RAM after the completion of the first TCD. The new TCD has its start bit already set and immediately starts the required write to the I^2C IBCR register to disable the DMAEN bit. This TCD also has scatter-gather activated and is programmed to reload the initial TCD upon completion, bringing the

system back into a ready-for-I^2C-transfer state. The advantage over the two other solutions is that this does not require CPU intervention or a second DMA channel. This comes at the cost of 64 bytes RAM (two TCDs), some system bus transfer overhead, and a little increase in application code complexity. On the system level, no higher priority DMA requests must occur during the scatter-gather process because those can result in a slow reaction.

Example latencies for a 32 MHz system with a full speed 32-bit AHB bus and an I^2C connected via half speed IPI bus:

- Accessing the I^2C from the DMA controller via IPI bus typically requires four cycles (consecutive accesses to the I^2C could be faster):

$$4 \times T_{IPI} = 4 / 16\ MHz = 250\ ns \qquad \textit{Eqn. 32-6}$$

- Reloading a new TCD (8 × 32 bit) via AHB to the DMA controller (scatter/gather process):

$$8 \times T_{AHB} = 8 / 32\ MHz = 250\ ns \qquad \textit{Eqn. 32-7}$$

With the DMA scatter-gather process, the required IBCR access can be done in 0.5 µs, leaving a large margin of 19.5 µs for additional system delays. The slow reaction case can be prevented in this way. The system user must decide which usage model suits his overall requirements best.

Chapter 33
Cross Triggering Unit (CTU)

33.1 Introduction

The Cross Triggering Unit (CTU) is a collection of 9-bit down-counters with an exponential prescaler able to generate the trigger event for the ADC conversion. The delayed trigger event can be a combination of unified channel flags/triggering events connected to different timers (eMIOS/PIT) present in the system. The 7-bit channel number is provided to the ADC. This channel number is used to communicate which particular channel has to be converted.

NOTE

The CTU is not implemented on the PXN20.

33.2 Main Features

- 9-bit down-counters counting from a programmable start value to 0
- Different counters associated with different channel groups
- Channel group is defined based on PWM channel clock
- Different delay value for each eMIOS flag/PIT event
- 4-bit programmable exponential prescaler: f_{CK_TIM} divided by 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024
- Single cycle delayed trigger output. The trigger output is a combination of 33 input flags/events connected to different timers in the system.
- Maskable interrupt generation whenever a trigger output is generated
- Event configuration registers dedicated to each UC flag/triggering event storing the values of a channel number to be provided for ATD conversion, CLR_FLAG bit, counter selection bits, delay value selection bits and trigger masking for each UC flag/triggering event
- Acknowledgment signal to eMIOS/PIT for clearing the flag
- Synchronization with ADC to avoid collision

33.3 Block Diagram

The block diagram is shown in Figure 33-1.

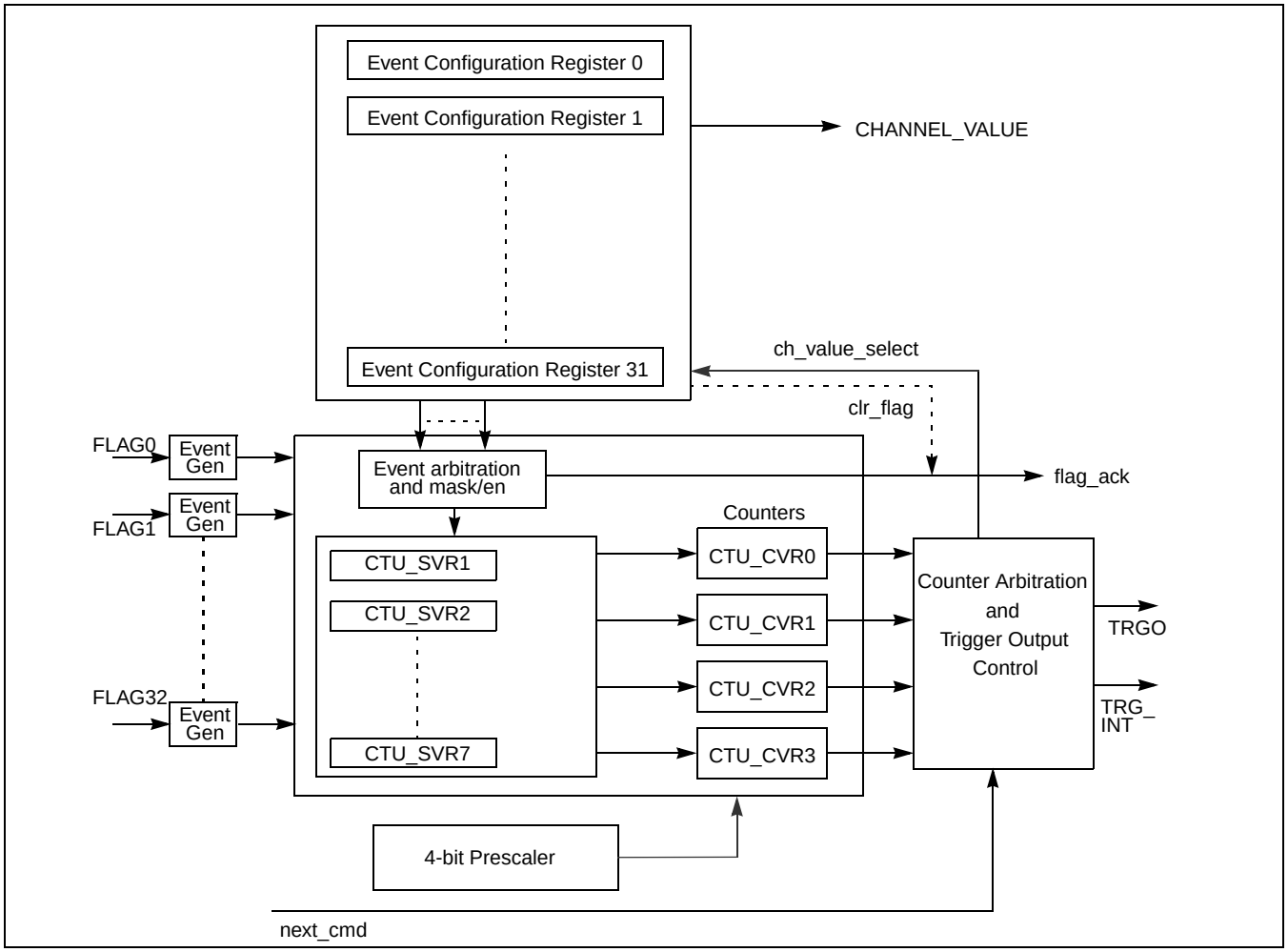


Figure 33-1. Cross Triggering Unit Block Diagram

33.4 Memory Map and Register Description

This section provides a detailed description of all CTU registers.

33.4.1 Module Memory Map

The CTU registers are listed in Table 33-1.

Table 33-1. CTU Memory Map

Offset from CTU_BASE 0xFFFD_8000	Register	Access	Reset Value	Section/Page	Size
0x0000	CTU_CSR – Control Status Register	R/W[1]	0x0000_0000	33.4.1.1/33-4	32
0x0004	CTU_SVR1 – Start Value Register 1	R/W[1]	0x0000_0000	33.4.1.2/33-5	32
0x0008	CTU_SVR2 – Start Value Register 2	R/W[1]	0x0000_0000	33.4.1.2/33-5	32
0x000C	CTU_SVR3 – Start Value Register 3	R/W[1]	0x0000_0000	33.4.1.2/33-5	32

Table 33-1. CTU Memory Map

Offset from CTU_BASE 0xFFFD_8000	Register	Access	Reset Value	Section/Page	Size
0x0010	CTU_SVR4 – Start Value Register 4	R/W[1]	0x0000_0000	33.4.1.2/33-5	32
0x0014	CTU_SVR5 – Start Value Register 5	R/W[1]	0x0000_0000	33.4.1.2/33-5	32
0x0018	CTU_SVR6 – Start Value Register 6	R/W[1]	0x0000_0000	33.4.1.2/33-5	32
0x001C	CTU_SVR7 – Start Value Register 7	R/W[1]	0x0000_0000	33.4.1.2/33-5	32
0x0020	CTU_CVR0 – Current Value Register 0	RO	0x0000_0000	33.4.1.3/33-5	32
0x0024	CTU_CVR1 – Current Value Register 1	RO	0x0000_0000	33.4.1.3/33-5	32
0x0028	CTU_CVR2 – Current Value Register 2	RO	0x0000_0000	33.4.1.3/33-5	32
0x002C	CTU_CVR3 – Current Value Register 3	RO	0x0000_0000	33.4.1.3/33-5	32
0x0030	CTU_EVTCFGR0[2] – Event Configuration Register 0	R/W[1]	0x0000_0000	33.4.1.4/33-6	32
0x0034	CTU_EVTCFGR1[2] – Event Configuration Register 1	R/W[1]	0x0000_0000	33.4.1.4/33-6	32
0x0038	CTU_EVTCFGR2[2] – Event Configuration Register 2	R/W1	0x0000_0000	33.4.1.4/33-6	32
0x003C	CTU_EVTCFGR3[2] – Event Configuration Register 3	R/W[1]	0x0000_0000	33.4.1.4/33-6	32
0x0040	CTU_EVTCFGR4[2] – Event Configuration Register 4	R/W[1]	0x0000_0000	33.4.1.4/33-6	32
0x0044	CTU_EVTCFGR5[2] – Event Configuration Register 5	R/W[1]	0x0000_0000	33.4.1.4/33-6	32
0x0048	CTU_EVTCFGR6[2] – Event Configuration Register 6	R/W[1]	0x0000_0000	33.4.1.4/33-6	32
0x004C	CTU_EVTCFGR7[2] – Event Configuration Register 7	R/W[1]	0x0000_0000	33.4.1.4/33-6	32
0x0050	CTU_EVTCFGR8[2] – Event Configuration Register 8	R/W[1]	0x0000_0000	33.4.1.4/33-6	32
0x0054	CTU_EVTCFGR9[2] – Event Configuration Register 9	R/W[1]	0x0000_0000	33.4.1.4/33-6	32
0x0058	CTU_EVTCFGR10[2] – Event Configuration Register 10	R/W[1]	0x0000_0000	33.4.1.4/33-6	32
0x005C	CTU_EVTCFGR11[2] – Event Configuration Register 11	R/W[1]	0x0000_0000	33.4.1.4/33-6	32
0x0060	CTU_EVTCFGR12[2] – Event Configuration Register 12	R/W[1]	0x0000_0000	33.4.1.4/33-6	32
0x0064	CTU_EVTCFGR13[2] – Event Configuration Register 13	R/W[1]	0x0000_0000	33.4.1.4/33-6	32
0x0068	CTU_EVTCFGR14[2] – Event Configuration Register 14	R/W[1]	0x0000_0000	33.4.1.4/33-6	32
0x006C	CTU_EVTCFGR15[2] – Event Configuration Register 15	R/W[1]	0x0000_0000	33.4.1.4/33-6	32
0x0070	CTU_EVTCFGR16[2] – Event Configuration Register 16	R/W[1]	0x0000_0000	33.4.1.4/33-6	32
0x0074	CTU_EVTCFGR17[2] – Event Configuration Register 17	R/W[1]	0x0000_0000	33.4.1.4/33-6	32
0x0078	CTU_EVTCFGR18[2] – Event Configuration Register 18	R/W[1]	0x0000_0000	33.4.1.4/33-6	32
0x007C	CTU_EVTCFGR19[2] – Event Configuration Register 19	R/W[1]	0x0000_0000	33.4.1.4/33-6	32
0x0080	CTU_EVTCFGR20[2] – Event Configuration Register 20	R/W[1]	0x0000_0000	33.4.1.4/33-6	32
0x0084	CTU_EVTCFGR21[2] – Event Configuration Register 21	R/W[1]	0x0000_0000	33.4.1.4/33-6	32
0x0088	CTU_EVTCFGR22[2] – Event Configuration Register 22	R/W[1]	0x0000_0000	33.4.1.4/33-6	32
0x008C	CTU_EVTCFGR23[2] – Event Configuration Register 23	R/W[1]	0x0000_0000	33.4.1.4/33-6	32
0x0090	CTU_EVTCFGR24[2] – Event Configuration Register 24	R/W[1]	0x0000_0000	33.4.1.4/33-6	32

Table 33-1. CTU Memory Map

Offset from CTU_BASE 0xFFFD_8000	Register	Access	Reset Value	Section/Page	Size
0x0094	CTU_EVTCFGR25[2] – Event Configuration Register 25	R/W[1]	0x0000_0000	33.4.1.4/33-6	32
0x0098	CTU_EVTCFGR26[2] – Event Configuration Register 26	R/W[1]	0x0000_0000	33.4.1.4/33-6	32
0x009C	CTU_EVTCFGR27[2] – Event Configuration Register 27	R/W[1]	0x0000_0000	33.4.1.4/33-6	32
0x00A0	CTU_EVTCFGR28[2] – Event Configuration Register 28	R/W[1]	0x0000_0000	33.4.1.4/33-6	32
0x00A4	CTU_EVTCFGR29[2] – Event Configuration Register 29	R/W[1]	0x0000_0000	33.4.1.4/33-6	32
0x00A8	CTU_EVTSELR30[2] – Event Configuration Register 30	R/W[1]	0x0000_0000	33.4.1.4/33-6	32
0x00AC	CTU_EVTSELR31[2] – Event Configuration Register 31	R/W[1]	0x0000_0000	33.4.1.4/33-6	32
0x00B0	CTU_EVTSELR32[3] – Event Configuration Register 32	R/W[1]	0x0000_0000	33.4.1.4/33-6	32
0x00B4–0x3FFF	Reserved				

1 Some bits are read-only.

2 For eMIOS channels 0 – 31.

3 For PIT3.

33.4.1.1 Control Status Register (CTU_CSR)

Offset: CTU_BASE + 0x0000 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	TRGIEN	TRGI	0	0	PRESC_CONF			
W										w1c						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 33-2. Control Status Register (CTU_CSR)

Table 33-2. CTU_CSR Register Field Descriptions

Bit	Description
TRGIEN	Trigger Interrupt Request Enable 0 Trigger interrupt request disabled. 1 Trigger interrupt request enabled. A request is generated if the TRGI flag is set.
TRGI	Trigger Interrupt Flag. This flag is set by hardware when the trigger output request is generated after a valid input event is detected. It is cleared by software. 0 No trigger output request. 1 Trigger output request generated.

Table 33-2. CTU_CSR Register Field Descriptions (continued)

Bit	Description
PRESC_ CONF	Prescaler Configuration. The counter clock is derived from the system clock by dividing it by 2^0 to 2^{10} depending on the prescaler configuration bits. The clock division coding is as follows: 0000 Clock divided by 1 (no division). 0001 Clock divided by 2. 0010 Clock divided by 4. 0011 Clock divided by 8. 0100 Clock divided by 16. 0101 Clock divided by 32. 0110 Clock divided by 64. 0111 Clock divided by 128. 1000 Clock divided by 256. 1001 Clock divided by 512. 1010 Clock divided by 1024. 1011 – 1111 Clock divided by 1(no division).

33.4.1.2 Start Value Register (CTU_SVR*n*)

The CTU_SVR*n* registers contain the start values to be loaded into the CTU_CVR*m* registers.

Offset: CTU_BASE + 0x0004 (CTU_SVR1), 0x0008 (CTU_SVR2), 0x000C(CTU_SVR3), 0x0010 (CTU_SVR4), 0x0014 (CTU_SVR5), 0x0018 (CTU_SVR6), 0x001C (CTU_SVR7)

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	SV[8:0]								
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 33-3. Start Value Register (CTU_SVR*n*)

Table 33-4. CTU_SVR*n* Register Field Descriptions

Bit	Description
SV[8:0]	Start Value. These bits store the start value of the counting to be loaded into the current value register whenever an event mapped to this start value register is serviced.

33.4.1.3 Current Value Register (CTU_CVR*m*)

The CTU_CVR*m* registers contain the current count value.

Offset: CTU_BASE + 0x0020 (CTU_CVR0) 0x0028 (CTU_CVR2)
0x0024 (CTU_CVR1) 0x002C (CTU_CVR3)
Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	CV[8:0]								
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 33-5. Current Value Register (CTU_CVR*n*)

Table 33-6. CTU_CVR*m* Register Field Descriptions

Bit	Description
CV[8:0]	Current Value. These bits contain the current value of the counter. The counter starts counting from the start value loaded from corresponding start value register down to 0x000 as soon as a valid input event is detected.

33.4.1.4 Event Configuration Register (CTU_EVTCFGR*n*)

Event configuration registers 0 – 31 are associated with eMIOS channels 0 – 31. Event configuration register 32 is associated with PIT3.

Offset: CTU_BASE + 0x0030–0x00B0
Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	TM	0	COUNT_GROUP		0	DELAY_INDEX			CLR_FLAG	0	CHANNEL_VALUE					
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 33-7. Event Configuration Register (CTU_EVTCFGR*n*)

Table 33-8. CTU_EVTCFGR*n* Register Field Descriptions

Bit	Description
TM	Trigger Mask. 0 Trigger masked. 1 Trigger enabled.

Table 33-8. CTU_EVTCFGR*n* Register Field Descriptions (continued)

Bit	Description
COUNT_GROUP	Counter Group. 00 Counter 0 is associated with the particular event. 01 Counter 1 is associated with the particular event. 10 Counter 2 is associated with the particular event. 11 Counter 3 is associated with the particular event.
DELAY_INDEX	Delay Index. 000 No delay is provided. 001 Counter is loaded with value stored in CTU_SVR1. 010 Counter is loaded with value stored in CTU_SVR2 011 Counter is loaded with value stored in CTU_SVR3 100 Counter is loaded with value stored in CTU_SVR4 101 Counter is loaded with value stored in CTU_SVR5 110 Counter is loaded with value stored in CTU_SVR6 111 Counter is loaded with value stored in CTU_SVR7
CLR_FLAG[1]	To provide flag_ack through software. 0 Flag_ack is dependent on flag servicing. 1 Flag_ack is forced to '1' for the particular event.
CHANNEL_VALUE	Channel value to be provided to ADC.

1 This bit implementation is generic based and implemented only for inputs mapped to PIT event flags.

These registers contain the channel value signifying which channel needs to be ATD converted, the CLR_FLAG used to clear ocmp flag through software, the counter group that specifies the clock group to which input channel is associated, the delay selection bits that have to be loaded into the counters and the mask/enable for the event.

The CLR_FLAG bit has to be used cautiously as setting this bit may result in loss of events.

The event input can be masked by setting trigger mask bit of CTU_EVTCFGR*n* register to '0'.

33.5 Functional Description

The CTU is used to generate a trigger output for the conversion and provide the channel to be converted. The trigger output is a combination of gated events of different eMIOS/PIT flags with configurable delays. The trigger output is a single cycle pulse used to trigger an ADC conversion of the channel number provided by CTU.

Each event has a dedicated configuration register (CTU_EVTCFGR*n*). These registers store a channel number which is used to communicate which channel needs to be converted, the counter group, the start value selection bits and masking bit for that particular event.

The CTU interfaces between the eMIOS/PIT and the ADC, and convert the events generated by the eMIOS into ADC conversion requests.

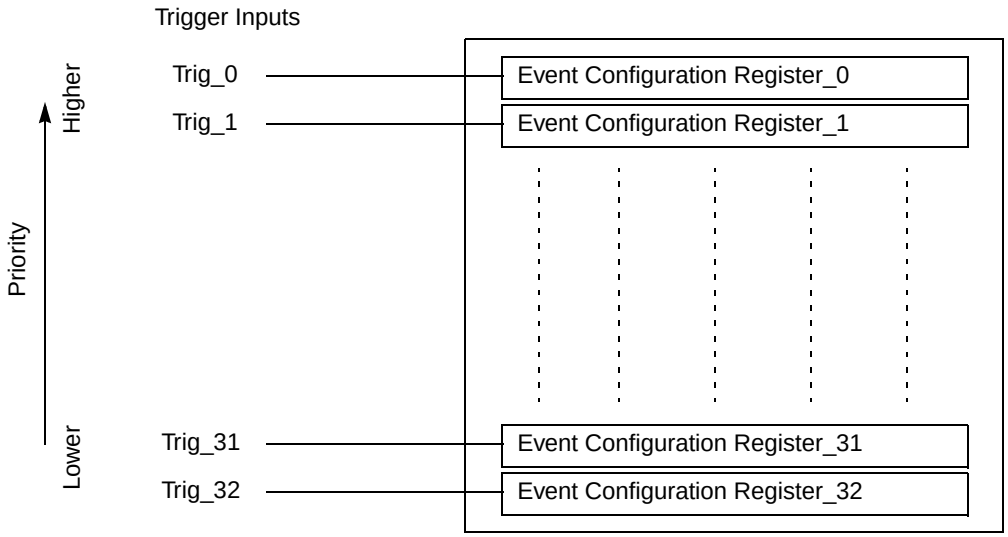


Figure 33-3. CTU Trigger Sources Allocation

All the flags can be divided in to four different counter groups which signify which PWM channel clock group the particular event is generated. The four counters can run in parallel and the trigger output is generated for the counter expiring first. If more than one counter reaches zero at the same time, the lower index counter has higher priority and the corresponding channel value is provided to the ADC.

Each trigger input from the CTU is connected to the Event Trigger signal of an eMIOS or PIT channel. The assignment between eMIOS/PIT outputs and CTU trigger inputs is defined in Table 33-9.

The eMIOS signal “FLAG” is used in DMA mode to interface with the trigger input of the CTU. The CTU resets the FLAG signal once the ADC conversion request has been completed.

NOTE

The eMIOS channels can either be used with eDMA or CTU. They cannot be used with eDMA and CTU at the same time.

Table 33-9. CTU Trigger Sources

Trigger Number	Module	Source
0	eMIOS	Channel_0
1	eMIOS	Channel_1
2	eMIOS	Channel_2
3	eMIOS	Channel_3
4	eMIOS	Channel_4
5	eMIOS	Channel_5
6	eMIOS	Channel_6
7	eMIOS	Channel_7
8	eMIOS	Channel_8

Table 33-9. CTU Trigger Sources (continued)

Trigger Number	Module	Source
9	eMIOS	Channel_9
10	eMIOS	Channel_10
11	eMIOS	Channel_11
12	eMIOS	Channel_12
13	eMIOS	Channel_13
14	eMIOS	Channel_14
15	eMIOS	Channel_15
16	eMIOS	Channel_16
17	eMIOS	Channel_17
18	eMIOS	Channel_18
19	eMIOS	Channel_19
20	eMIOS	Channel_20
21	eMIOS	Channel_21
22	eMIOS	Channel_22
23	eMIOS	Channel_23
24	eMIOS	Channel_24
25	eMIOS	Channel_25
26	eMIOS	Channel_26
27	eMIOS	Channel_27
28	eMIOS	Channel_28
29	eMIOS	Channel_29
30	eMIOS	Channel_30
31	eMIOS	Channel_31
32	PIT	PIT_4

Whenever a flag is set in a particular counter group, the corresponding counter loads the value from one of the start value registers depending on the delay selection bits. An acknowledgement signal is sent to eMIOS/PIT to clear the flag. In case more than one flag is set in the same counter group, the lower index flag is given priority and the corresponding delay value is loaded into the counter.

The acknowledgment signal can be forced to ‘1’ by setting the CLR_FLAG bit of the CTU_EVTCFGR*n* register. These bits are implemented for only those input flags to which PIT flags are connected. The purpose to provide these bits is to have the option of clearing PIT flags by software.

In summary, two levels of arbitration are done before the channel number and trigger are provided to the ADC:

- First level of arbitration is done between all events associated to one counter. The lowest index channel has the highest priority.
- Second level of arbitration is done between counters. The lowest index counter has the highest priority.

33.5.1 Pending Request

In case multiple events occur in the same counter group, the sequence below is followed:

1. The lowest index event is given priority, and a corresponding delay is loaded into the counter.
2. The counter starts counting down. Other events remain pending.
3. The eMIOS/PIT flag is cleared for the event being serviced.
4. When the counter reaches zero, ADC conversion is triggered, provided that the previous conversion (if any) is completed and arbitration grants the trigger to this counter.
5. The counter is loaded with the delay value corresponding to next pending flag and the sequence follows. In case no more flags are pending, the counter resets.

The counters are also reset if the CTU halt bit (bit 9) in the SIU_HLT1 register is set.

33.5.2 Counter

The counters are 9-bit down-counters which are triggered by the logical OR output of the input events for that counter group. A particular counter can only be triggered if some unmasked event is pending in that counter group. As soon as a valid input event is detected, the current value register CTU_CVR*m* (i.e., counter) is loaded with the corresponding start value for that particular event and the counter starts counting down. The start value for the particular event is based on the delay selection bits of the corresponding CTU_EVTCFGR register. The counter stops automatically when the count value reaches ‘0’ and provides the ADC trigger and corresponding channel number, provided that it wins the arbitration and the previous ADC conversion (if any) is over.

In case some flags are pending in the counter group and they are not masked, the counter is reloaded with a delay value corresponding to the lowest index pending flag and starts counting down again. If no valid flag is set for the counter group, the counter remains in reset state.

The counters are in the following states after each reset condition:

- System reset — Reset (all zeros)
- Power On Reset — Reset (all zeros)
- Wake-up from low power Sleep mode — Reset (all zeros)

33.5.3 Prescaler

The counter clock is the prescaler output. To offer a wide counting period range, the prescaler allows to divide the clock by 2^0 to 2^{10} depending on the prescaler configuration (PRESC_CONF) bits of the CTU_CSR register.

The PRESC_CONF bits of register CTU_CSR must not be modified while the counters are running. The counter behavior is not guaranteed if this rule is not respected. Also, it is recommended to program the CTU_SVR*n* registers only when counters are not running.

33.5.4 Trigger Interrupt Request

When a request for trigger output is generated, the flag TRGI in the CTU_CSR register is set. An interrupt is generated if the TRGIEN bit in the CTU_CSR register is set. If this condition is false, the interrupt remains pending to be issued as soon as it is enabled. The interrupt status flag can be cleared by writing '1' to the TRGI bit.

33.5.5 Halt Request

Whenever a halt mode entry request is generated (CTU halt bit in the SIU_HLT0 register is set), the counters are reset. Setting this bit also turns off the clock to the module, shutting it down.

33.5.6 Channel Value

The channel value stored in an event configuration register is demultiplexed to 7 bits and then provided to the ADC. The mapping of channel number value to the corresponding ADC channel is provided in Table 33-10.

Table 33-10. Channel Number Value Mapping

ADC Channel	CTU Channel
Channel 0–15	Channel 0–15
Channel 16–31	— not mapped —
Channel 32–47	Channel 16–31
Channel 48–63	— not mapped —
Channel 64–95	Channel 32–64

So while programming an event configuration register, this mapping has to be taken care of, e.g., if the channel value of any event configuration register is programmed to 16, it actually corresponds to channel 32 of ADC and conversion occurs for this channel.

Chapter 34
Analog-to-Digital Converter (ADC)

34.1 Introduction

The ADC module contains advanced features for normal, injected, and triggered injected conversion, along with offset cancellation and offset refresh control, and supports the interface to the Cross Triggering Unit (CTU). The ADC contains user-configurable sampling and conversion times with a clock prescaler unit to generate the ADC clock from the clock provided to the ADC digital interface. The ADC contains "analog watchdogs" for comparing values of the converted data against user programmed thresholds and interrupt generation based on threshold violation by the converted data.

NOTE

The CTU is not implemented on the PXN20.

34.1.1 Block Diagram

Figure 34-1 shows the ADC block diagram.

It can be configured to generate sampling signals for as many as 96 multiplexed analog input channels. There are three types of input channels: internal channels (group 0 and 1), and external channels (group 2).

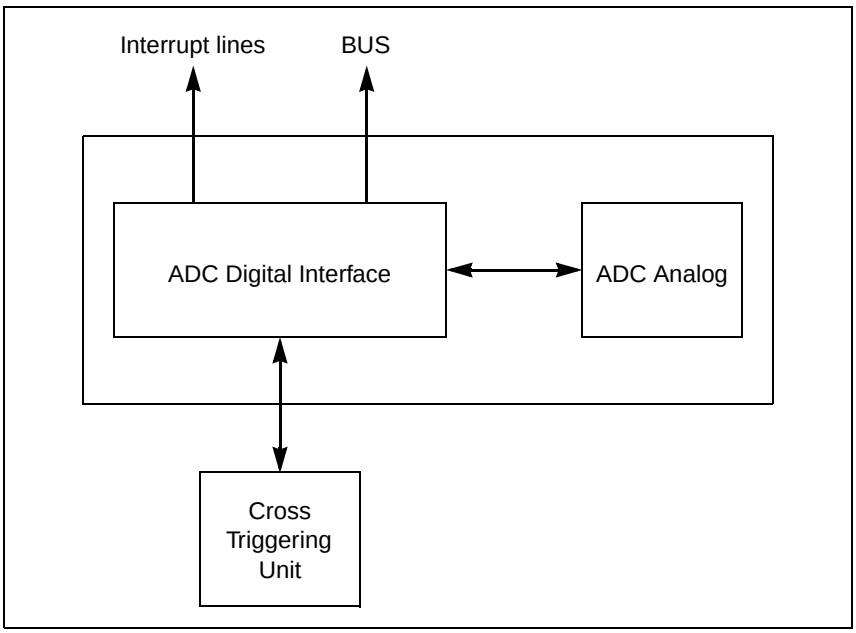


Figure 34-1. ADC Block Diagram

34.1.2 Features

The ADC digital interface provides the following features:

- 10-bit resolution
- As many as 64 internal channels
  - 16 high precision
  - 48 normal precision
- As many as 32 additional external channels supported
- Three different sampling and conversion time registers, one for each group of input channels
- Conversions on external channels have all the same features as the internal ones
- External decode signals (3 bits) for external analog mux
- Conversion speeds as short as 1 µs (1 MHz)
- Supports current injection ±1 ma
- One-shot/scan modes
- Chain injection mode
- Triggered injection
- Presampling
- Offset cancellation and offset refresh control
- External start feature
- Power down mode
- Two different abort features allow aborting either a single-channel conversion or chain conversion
- As many as 96 data registers for storing the converted data. Some conversion information as mode of operation (normal, injected, or CTU) is associated with data value.
- Configurable analog watchdog channels controlled by the generic parameter num_watchdog
- Alternate analog thresholds
- Auto clock off feature
- Clock prescaler (ipg_clk divided by 2)
- CTU trigger mode
- Big Endian

34.2 External Signals

The ADC supports two types of external signals: power signals and analog input signals. See Chapter 3, Signal Description. Table 3-1 describes the analog input signals and Table 3-2 describes the power signals.

NOTE

In Chapter 3, Signal Description, AN[0:63] = internal channels.

34.3 Memory Map and Register Definition

This section provides memory maps and detailed descriptions of all registers. Data written to or read from reserved areas of the memory map is undefined.

34.3.1 ADC Memory Map

This section provides the memory map for the ADC.

Table 34-1. ADC Memory Map

Offset from ADC_BASE (ADC_A= 0xFFF8_0000)	Register	Access	Reset Value	Section/Page	Size
0x0000	MCR — Main Configuration Register	R/W[1]	0x0000_0001	34.3.2.1/34-8	32
0x0004	MSR — Main Status Register	RO	0x0000_0001	34.3.2.2/34-10	32
0x0008–0x000C	Reserved				
0x0010	ISR — Interrupt Status Register	R/W[1]	0x0000_0000	34.3.2.3/34-11	32
0x0014	CEOCFR0 — Channel Pending Register 0	R/W[1]	0x0000_0000	34.3.2.4/34-12	32
0x0018	CEOCFR1 — Channel Pending Register 1	R/W[1]	0x0000_0000	34.3.2.5/34-13	32
0x001C	CEOCFR2 — Channel Pending Register 2	R/W[1]	0x0000_0000	34.3.2.6/34-13	32
0x0020	IMR — Interrupt Mask Register	R/W[1]	0x0000_0000	34.3.2.7/34-14	32
0x0024	CIMR0 — Channel Interrupt Mask Register 0	R/W[1]	0x0000_0000	34.3.2.8/34-15	32
0x0028	CIMR1 — Channel Interrupt Mask Register 1	R/W[1]	0x0000_0000	34.3.2.9/34-15	32
0x002C	CIMR2 — Channel Interrupt Mask Register 2	R/W[1]	0x0000_0000	34.3.2.10/34-16	32
0x0030	WTISR — Watchdog Interrupt Threshold Register	R/W[1]	0x0000_0000	34.3.2.11/34-16	32
0x0034	WTIMR — Watchdog Interrupt Threshold Mask Register	R/W[1]	0x0000_0000	34.3.2.12/34-17	32
0x0038–0x003C	Reserved				
0x0040	DMAE — DMA Enable Register	R/W[1]	0x0000_0000	34.3.2.13/34-18	32
0x0044	DMAR0 — DMA Channel Select Register 0	R/W[1]	0x0000_0000	34.3.2.14/34-18	32
0x0048	DMAR1 — DMA Channel Select Register 1	R/W[1]	0x0000_0000	34.3.2.15/34-19	32
0x004C	DMAR2 — DMA Channel Select Register 2	R/W[1]	0x0000_0000	34.3.2.16/34-19	32
0x0050	TRC0 — Threshold Control Register 0	R/W[1]	0x0000_0000	34.3.2.17/34-20	32
0x0054h	TRC1 — Threshold Control Register 1	R/W[1]	0x0000_0000	34.3.2.17/34-20	32
0x0058	TRC2 — Threshold Control Register 2	R/W[1]	0x0000_0000	34.3.2.17/34-20	32
0x005C	TRC3 — Threshold Control Register 3	R/W[1]	0x0000_0000	34.3.2.17/34-20	32
0x0060	THRHLR0 — Threshold Register 0	R/W[1]	0x0FFF_0000	34.3.2.18/34-21	32
0x0064	THRHLR1 — Threshold Register 1	R/W[1]	0x0FFF_0000	34.3.2.18/34-21	32
0x0068	THRHLR2 — Threshold Register 2	R/W[1]	0x0FFF_0000	34.3.2.18/34-21	32

Table 34-1. ADC Memory Map (continued)

Offset from ADC_BASE (ADC_A= 0xFFF8_0000)	Register	Access	Reset Value	Section/Page	Size
0x006C	THRHLR3 — Threshold Register 3	R/W[1]	0x0FFF_0000	34.3.2.18/34-21	32
0x0070–0x007C	Reserved				
0x0080	PSCR — Presampling Control Register	R/W[1]	0x0000_0000	34.3.2.19/34-22	32
0x0084	PSR0 — Presampling Register 0	R/W[1]	0x0000_0000	34.3.2.20/34-22	32
0x0088	PSR1 — Presampling Register 1	R/W[1]	0x0000_0000	34.3.2.21/34-23	32
0x008C	PSR2 — Presampling Register 2	R/W[1]	0x0000_0000	34.3.2.22/34-23	32
0x0090	Reserved				
0x0094	CTR0 — Conversion Timing Register 0	R/W[1]	0x0000_0203	34.3.2.23/34-24	32
0x0098	CTR1— Conversion Timing Register 1	R/W[1]	0x0000_0203	34.3.2.24/34-24	32
0x009C	CTR2 — Conversion Timing Register 2	R/W[1]	0x0000_0203	34.3.2.25/34-25	32
0x00A0	Reserved				
0x00A4	NCMR0 — Normal Conversion Mask Register 0	R/W[1]	0x0000_0000	34.3.2.26/34-33	32
0x00A8	NCMR1 — Normal Conversion Mask Register 1	R/W[1]	0x0000_0000	34.3.2.27/34-34	32
0x00AC	NCMR2 — Normal Conversion Mask Register 2	R/W[1]	0x0000_0000	34.3.2.28/34-34	32
0x00B0	Reserved				
0x00B4	JCMR0 — Injected Conversion Mask Register 0	R/W[1]	0x0000_0000	34.3.2.29/34-35	32
0x00B8	JCMR1 — Injected Conversion Mask Register 1	R/W[1]	0x0000_0000	34.3.2.30/34-35	32
0x00BC	JCMR2 — Injected Conversion Mask Register 2	R/W[1]	0x0000_0000	34.3.2.31/34-36	32
0x00C0	OFFWR — Offset Word Register	R/W[1]	0x0000_0000	34.3.2.32/34-36	32
0x00C4	DSDR — Decode Signals Delay Register	R/W[1]	0x0000_0000	34.3.2.33/34-37	32
0x00C8	PDEDR — Power Down Exit Delay Register	R/W[1]	0x0000_0000	34.3.2.34/34-38	32
0x00CC – 0x00FC	Reserved				
0x0100	PRECDATAREG0 — Channel 0 Data Register	RO	0x0000_0000	34.3.2.35/34-38	32
0x0104	PRECDATAREG1 — Channel 1 Data Register	RO	0x0000_0000	34.3.2.35/34-38	32
0x0108	PRECDATAREG2 — Channel 2 Data Register	RO	0x0000_0000	34.3.2.35/34-38	32
0x010C	PRECDATAREG3 — Channel 3 Data Register	RO	0x0000_0000	34.3.2.35/34-38	32
0x0110	PRECDATAREG4 — Channel 4 Data Register	RO	0x0000_0000	34.3.2.35/34-38	32
0x0114	PRECDATAREG5 — Channel 5 Data Register	RO	0x0000_0000	34.3.2.35/34-38	32
0x0118	PRECDATAREG6 — Channel 6 Data Register	RO	0x0000_0000	34.3.2.35/34-38	32
0x011C	PRECDATAREG7 — Channel 7 Data Register	RO	0x0000_0000	34.3.2.35/34-38	32
0x0120	PRECDATAREG8 — Channel 8 Data Register	RO	0x0000_0000	34.3.2.35/34-38	32

Table 34-1. ADC Memory Map (continued)

Offset from ADC_BASE (ADC_A= 0xFFF8_0000)	Register	Access	Reset Value	Section/Page	Size
0x0124	PRECDATAREG9 — Channel 9 Data Register	RO	0x0000_0000	34.3.2.35/34-38	32
0x0128	PRECDATAREG10 — Channel 10 Data Register	RO	0x0000_0000	34.3.2.35/34-38	32
0x012C	PRECDATAREG11 — Channel 11 Data Register	RO	0x0000_0000	34.3.2.35/34-38	32
0x0130	PRECDATAREG12 — Channel 12 Data Register	RO	0x0000_0000	34.3.2.35/34-38	32
0x0134	PRECDATAREG13 — Channel 13 Data Register	RO	0x0000_0000	34.3.2.35/34-38	32
0x0138	PRECDATAREG14 — Channel 14 Data Register	RO	0x0000_0000	34.3.2.35/34-38	32
0x013C	PRECDATAREG15 — Channel 15 Data Register	RO	0x0000_0000	34.3.2.35/34-38	32
0x0140	PRECDATAREG16 — Channel 16 Data Register	RO	0x0000_0000	34.3.2.35/34-38	32
0x0144	PRECDATAREG17 — Channel 17 Data Register	RO	0x0000_0000	34.3.2.35/34-38	32
0x0148	PRECDATAREG18 — Channel 18 Data Register	RO	0x0000_0000	34.3.2.35/34-38	32
0x014C	PRECDATAREG19 — Channel 19 Data Register	RO	0x0000_0000	34.3.2.35/34-38	32
0x0150	PRECDATAREG20 — Channel 20 Data Register	RO	0x0000_0000	34.3.2.35/34-38	32
0x0154	PRECDATAREG21 — Channel 21 Data Register	RO	0x0000_0000	34.3.2.35/34-38	32
0x0158	PRECDATAREG22 — Channel 22 Data Register	RO	0x0000_0000	34.3.2.35/34-38	32
0x015C	PRECDATAREG23 — Channel 23 Data Register	RO	0x0000_0000	34.3.2.35/34-38	32
0x0160	PRECDATAREG24 — Channel 24 Data Register	RO	0x0000_0000	34.3.2.35/34-38	32
0x0164	PRECDATAREG25 — Channel 25 Data Register	RO	0x0000_0000	34.3.2.35/34-38	32
0x0168	PRECDATAREG26 — Channel 26 Data Register	RO	0x0000_0000	34.3.2.35/34-38	32
0x016C	PRECDATAREG27 — Channel 27 Data Register	RO	0x0000_0000	34.3.2.35/34-38	32
0x0170	PRECDATAREG28 — Channel 28 Data Register	RO	0x0000_0000	34.3.2.35/34-38	32
0x0174	PRECDATAREG29 — Channel 29 Data Register	RO	0x0000_0000	34.3.2.35/34-38	32
0x0178	PRECDATAREG30 — Channel 30 Data Register	RO	0x0000_0000	34.3.2.35/34-38	32
0x017C	PRECDATAREG31 — Channel 31 Data Register	RO	0x0000_0000	34.3.2.35/34-38	32
0x0180	INTDATAREG0 — Channel 32 Data Register	RO	0x0000_0000	34.3.2.36/34-39	32
0x0184	INTDATAREG1 — Channel 33 Data Register	RO	0x0000_0000	34.3.2.36/34-39	32
0x0188	INTDATAREG2 — Channel 34 Data Register	RO	0x0000_0000	34.3.2.36/34-39	32
0x018C	INTDATAREG3 — Channel 35 Data Register	RO	0x0000_0000	34.3.2.36/34-39	32
0x0190	INTDATAREG4 — Channel 36 Data Register	RO	0x0000_0000	34.3.2.36/34-39	32
0x0194	INTDATAREG5 — Channel 37 Data Register	RO	0x0000_0000	34.3.2.36/34-39	32
0x0198	INTDATAREG6 — Channel 38 Data Register	RO	0x0000_0000	34.3.2.36/34-39	32
0x019C	INTDATAREG7 — Channel 39 Data Register	RO	0x0000_0000	34.3.2.36/34-39	32

Table 34-1. ADC Memory Map (continued)

Offset from ADC_BASE (ADC_A= 0xFFF8_0000)	Register	Access	Reset Value	Section/Page	Size
0x01A0	INTDATAREG8 — Channel 40 Data Register	RO	0x0000_0000	34.3.2.36/34-39	32
0x01A4	INTDATAREG9 — Channel 41 Data Register	RO	0x0000_0000	34.3.2.36/34-39	32
0x01A8	INTDATAREG10 — Channel 42 Data Register	RO	0x0000_0000	34.3.2.36/34-39	32
0x01AC	INTDATAREG11 — Channel 43 Data Register	RO	0x0000_0000	34.3.2.36/34-39	32
0x01B0	INTDATAREG12 — Channel 44 Data Register	RO	0x0000_0000	34.3.2.36/34-39	32
0x01B4	INTDATAREG13 — Channel 45 Data Register)	RO	0x0000_0000	34.3.2.36/34-39	32
0x01B8	INTDATAREG14 — Channel 46 Data Register	RO	0x0000_0000	34.3.2.36/34-39	32
0x01BC	INTDATAREG15 — Channel 47 Data Register	RO	0x0000_0000	34.3.2.36/34-39	32
0x01C0	INTDATAREG16 — Channel 48 Data Register	RO	0x0000_0000	34.3.2.36/34-39	32
0x01C4	INTDATAREG17 — Channel 49 Data Register	RO	0x0000_0000	34.3.2.36/34-39	32
0x01C8	INTDATAREG18 — Channel 50 Data Register	RO	0x0000_0000	34.3.2.36/34-39	32
0x01CC	INTDATAREG19 — Channel 51 Data Register	RO	0x0000_0000	34.3.2.36/34-39	32
0x01D0	INTDATAREG20 — Channel 52 Data Register	RO	0x0000_0000	34.3.2.36/34-39	32
0x01D4	INTDATAREG21 — Channel 53 Data Register	RO	0x0000_0000	34.3.2.36/34-39	32
0x01D8	INTDATAREG22 — Channel 54 Data Register	RO	0x0000_0000	34.3.2.36/34-39	32
0x01DC	INTDATAREG23 — Channel 55 Data Register	RO	0x0000_0000	34.3.2.36/34-39	32
0x01E0	INTDATAREG24 — Channel 56 Data Register	RO	0x0000_0000	34.3.2.36/34-39	32
0x01E4	INTDATAREG25 — Channel 57 Data Register	RO	0x0000_0000	34.3.2.36/34-39	32
0x01E8	INTDATAREG26 — Channel 58 Data Register	RO	0x0000_0000	34.3.2.36/34-39	32
0x01EC	INTDATAREG27 — Channel 59 Data Register	RO	0x0000_0000	34.3.2.36/34-39	32
0x01F0	INTDATAREG28 — Channel 60 Data Register	RO	0x0000_0000	34.3.2.36/34-39	32
0x01F4	INTDATAREG29 — Channel 61 Data Register	RO	0x0000_0000	34.3.2.36/34-39	32
0x01F8	INTDATAREG30 — Channel 62 Data Register	RO	0x0000_0000	34.3.2.36/34-39	32
0x01FC	INTDATAREG31 — Channel 63 Data Register	RO	0x0000_0000	34.3.2.36/34-39	32
0x0200	EXTDATAREG0 — Channel 64 Data Register	RO	0x0000_0000	34.3.2.37/34-39	32
0x0204	EXTDATAREG1 — Channel 65 Data Register	RO	0x0000_0000	34.3.2.37/34-39	32
0x0208	EXTDATAREG2 — Channel 66 Data Register	RO	0x0000_0000	34.3.2.37/34-39	32
0x020C	EXTDATAREG3 — Channel 67 Data Register	RO	0x0000_0000	34.3.2.37/34-39	32
0x0210	EXTDATAREG4 — Channel 68 Data Register	RO	0x0000_0000	34.3.2.37/34-39	32
0x0214	EXTDATAREG5 — Channel 69 Data Register	RO	0x0000_0000	34.3.2.37/34-39	32
0x0218	EXTDATAREG6 — Channel 70 Data Register	RO	0x0000_0000	34.3.2.37/34-39	32

Table 34-1. ADC Memory Map (continued)

Offset from ADC_BASE (ADC_A= 0xFFF8_0000)	Register	Access	Reset Value	Section/Page	Size
0x021C	EXTDATAREG7 — Channel 71 Data Register	RO	0x0000_0000	34.3.2.37/34-39	32
0x0220	EXTDATAREG8 — Channel 72 Data Register	RO	0x0000_0000	34.3.2.37/34-39	32
0x0224	EXTDATAREG9 — Channel 73 Data Register	RO	0x0000_0000	34.3.2.37/34-39	32
0x0228	EXTDATAREG10 — Channel 74 Data Register	RO	0x0000_0000	34.3.2.37/34-39	32
0x022C	EXTDATAREG11 — Channel 75 Data Register	RO	0x0000_0000	34.3.2.37/34-39	32
0x0230	EXTDATAREG12 — Channel 76 Data Register	RO	0x0000_0000	34.3.2.37/34-39	32
0x0234	EXTDATAREG13 — Channel 77 Data Register	RO	0x0000_0000	34.3.2.37/34-39	32
0x0238	EXTDATAREG14 — Channel 78 Data Register	RO	0x0000_0000	34.3.2.37/34-39	32
0x023C	EXTDATAREG15 — Channel 79 Data Register	RO	0x0000_0000	34.3.2.37/34-39	32
0x0240	EXTDATAREG16 — Channel 80 Data Register	RO	0x0000_0000	34.3.2.37/34-39	32
0x0244	EXTDATAREG17 — Channel 81 Data Register	RO	0x0000_0000	34.3.2.37/34-39	32
0x0248	EXTDATAREG18 — Channel 82 Data Register	RO	0x0000_0000	34.3.2.37/34-39	32
0x024C	EXTDATAREG19 — Channel 83 Data Register	RO	0x0000_0000	34.3.2.37/34-39	32
0x0250	EXTDATAREG20 — Channel 84 Data Register	RO	0x0000_0000	34.3.2.37/34-39	32
0x0254	EXTDATAREG21 — Channel 85 Data Register	RO	0x0000_0000	34.3.2.37/34-39	32
0x0258	EXTDATAREG22 — Channel 86 Data Register	RO	0x0000_0000	34.3.2.37/34-39	32
0x025C	EXTDATAREG23 — Channel 87 Data Register	RO	0x0000_0000	34.3.2.37/34-39	32
0x0260	EXTDATAREG24 — Channel 88 Data Register	RO	0x0000_0000	34.3.2.37/34-39	32
0x0264	EXTDATAREG25 — Channel 89 Data Register	RO	0x0000_0000	34.3.2.37/34-39	32
0x0268	EXTDATAREG26 — Channel 90 Data Register	RO	0x0000_0000	34.3.2.37/34-39	32
0x026C	EXTDATAREG27 — Channel 91 Data Register	RO	0x0000_0000	34.3.2.37/34-39	32
0x0270	EXTDATAREG28 — Channel 92 Data Register	RO	0x0000_0000	34.3.2.37/34-39	32
0x0274	EXTDATAREG29 — Channel 93 Data Register	RO	0x0000_0000	34.3.2.37/34-39	32
0x0278	EXTDATAREG30 — Channel 94 Data Register	RO	0x0000_0000	34.3.2.37/34-39	32
0x027C	EXTDATAREG31 — Channel 95 Data Register	RO	0x0000_0000	34.3.2.37/34-39	32
0x0280– 0x3FFF	Reserved				

[1] Some bits are read-only.

34.3.2 ADC Register Descriptions

This section lists the ADC registers in address order and describes the registers and their bit fields.

34.3.2.1 Main Configuration Register (MCR)

The MCR provides configuration settings for the ADC.

Address: ADC_BASE + 0x0000 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	OWREN	WLSIDE	MODE	EDGLEV	TRGEN	EDGE	XSTRTEN	NSTART	0	JTRGEN	JEDGE	JSTART	0	0	CTUEN	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	ADCLKSEL	ABORTCHAIN	ABORT	ACKO	OFFREFRESH	OFFCANC	0	0	PWDN
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Figure 34-2. Main Configuration Register (MCR)

Table 34-2. MCR Field Descriptions

Field	Description
OWREN	Overwrite enable. 0 Conversion data is discarded. 1 Conversion data is overwritten by a newer result.
WLSIDE	Write Left /Right aligned. 0 Conversion data is written right-aligned. 1 Conversion data is left-aligned (from 15 to (15 – resolution + 1)).
MODE	One Shot/Scan mode selection. 0 One Shot mode: configure the normal conversion of one chain. 1 Scan mode: configure continuous chain conversion mode. When the programmed chain conversion is finished, the chain conversion restarts immediately.
EDGLEV	Edge /Level selection for external start trigger. 0 Edge configuration for external trigger usage. 1 Level configuration for external trigger usage.
TRGEN	External trigger enable. This bit must be set to use external triggering to start a conversion. 0 An external trigger cannot be used to start a conversion. 1 An external trigger can start a conversion.

Table 34-2. MCR Field Descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>EDGE</td><td>Start trigger edge/ level detection. The following table shows the interaction between the EDGE bit and the TRGEN and EDGLEV bits.
<table>
<tr><th>TRGEN</th><th>EDGLEV</th><th>EDGE</th><th>Trigger Detection</th></tr>
<tr><td>0</td><td>n</td><td>n</td><td>External triggering disabled</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td>External trigger on falling edge of trigger</td></tr>
<tr><td>1</td><td>0</td><td>1</td><td>External trigger on rising edge of trigger</td></tr>
<tr><td>1</td><td>1</td><td>0</td><td>External trigger on low edge of trigger</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>External trigger on high edge of trigger</td></tr>
</table></td></tr>
<tr><td>XSTRTEN</td><td>External Start enable. This can be used in order to synchronize the START events of two ADCs.
0 START signal is disabled.
1 START signal is asserted when external START is detected.</td></tr>
<tr><td>NSTART</td><td>Normal Start conversion. Setting this bit starts the chain or scan conversion. Resetting this bit during scan mode causes the current chain conversion to finish, then stops the operation.
This bit stays high while the conversion is ongoing (or pending during injection mode).
0 Causes the current chain conversion to finish and stops the operation.
1 Starts the chain or scan conversion.</td></tr>
<tr><td>JTRGEN</td><td>Injection external trigger enable.
0 External trigger disabled for channel injection (injected conversion cannot be started using an external signal).
1 External trigger enabled for channel injection.</td></tr>
<tr><td>JEDGE</td><td>Injection trigger edge selection. Edge selection for external trigger, if JTRGEN = 1.
0 Selects falling edge for the external trigger.
1 Selects rising edge for the external trigger.</td></tr>
<tr><td>JSTART</td><td>Injection start. Setting this bit starts the configured injected analog channels to be converted by software. Resetting this bit has no effect, as the injected chain conversion cannot be interrupted.</td></tr>
<tr><td>CTUEN</td><td>Cross Triggering Unit enable.
0 The cross triggering unit is disabled and the triggered injected conversion cannot take place.
1 The cross triggering unit is enabled and the triggered injected conversion can take place.
Note: The CTU is not implemented on the PXN20.</td></tr>
<tr><td>ADCLKSEL</td><td>Analog clock frequency selector. When this bit is set, the AD_clk frequency is equal to ipg_clk frequency. Otherwise, the AD_clk frequency is half ipg_clk frequency. This bit can be written in Power Down mode only.
0 AD_clk frequency is half ipg_clk frequency.
1 AD_clk frequency is equal to ipg_clk frequency.</td></tr>
<tr><td>ABORTCHAIN</td><td>Abort chain. When this bit is set, the ongoing Chain Conversion is aborted. This bit is reset by hardware as soon as a new conversion is requested.
0 Conversion is not affected.
1 Aborts the ongoing chain conversion.</td></tr>
<tr><td>ABORT</td><td>Abort conversion. When this bit is set, the ongoing conversion is aborted and a new conversion is invoked. This bit is reset by hardware as soon as a new conversion is invoked.
0 Conversion is not affected.
1 Aborts the ongoing conversion.</td></tr>
</table>

Table 34-2. MCR Field Descriptions (continued)

Field	Description
ACKO	Auto clock off enable. If set, this bit enables the Auto clock off feature. See Section 34.4.10, Auto Clock Off Mode, for more information. 0 Auto clock off is disabled. 1 Auto clock off is enabled.
OFFREFRESH	Offset refresh enable. If set, this bit enables the offset refresh feature. 0 No offset refresh. 1 Offset refresh during idle mode when ADC is waiting for a new start of conversion.
OFFCANC	Offset Cancellation. This bit is reset to 0 when the offset cancellation ends. 0 No offset cancellation phase. 1 Offset cancellation phase before conversion.
PWDN	Power down enable. When this bit is set, the analog module is requested to enter Power Down mode. When ADC status is PWDN, resetting this bit starts ADC transition to IDLE mode. 0 ADC is in normal mode. 1 ADC has been requested to power down.

34.3.2.2 Main Status Register (MSR)

The MSR provides status bits for the ADC.

Address: ADC_BASE + 0x0004 Access: User read only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	N START	J ABORT	0	0	J START	0	0	0	CTU START
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CHADDR							0	0	0	ACKO	OFF REFR ESH	OFF CANC	ADCSTATUS		
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Figure 34-3. Main Status Register (MSR)

Table 34-3. MSR Field Descriptions

Field	Description
NSTART	This status bit indicates that a normal conversion is ongoing. 0 Normal conversion is not occurring now. 1 Normal conversion is occurring.
JABORT	This status bit indicates that an injected conversion has been aborted.This bit is reset when a new injected conversion starts. 0 New injected conversion has been started. 1 Injected conversion has been aborted.

Table 34-3. MSR Field Descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>JSTART</td><td>This status bit indicates that an injected conversion is ongoing.
0 Injected conversion is not occurring now.
1 Injected conversion is occurring.</td></tr>
<tr><td>CTUSTART</td><td>This status bit indicates that a CTU conversion is ongoing. This bit is set when a CTU trigger pulse is received and the CTU conversion starts. When CTU trigger mode is enabled, this bit is automatically reset when the conversion is completed. Otherwise, if Control Mode is enabled this bit is reset when the CTU is disabled (CTUEN = 0).
0 CTU conversion is not occurring now.
1 CTU conversion is occurring.
Note: The CTU is not implemented on the PXN20.</td></tr>
<tr><td>CHADDR</td><td>Channel under measure address. This bitfield indicates which channel (0 to 95) is under measure.</td></tr>
<tr><td>ACKO</td><td>Auto clock off enable. This status bit indicates whether the Auto clock off feature is activated.
0 Auto clock off feature is deactivated.
1 Auto clock off feature is activated.</td></tr>
<tr><td>OFFREFRESH</td><td>This status bit indicates that an offset refresh is ongoing.
0 No offset refresh.
1 Offset refresh during idle mode when ADC is waiting for a new start of conversion.</td></tr>
<tr><td>OFFCANC</td><td>This status bit indicates that an offset cancellation is ongoing.
0 Offset cancellation is not occurring.
1 Offset cancellation is occurring.</td></tr>
<tr><td>ADCSTATUS</td><td>This bitfield displays the ADC status, as follows:
<table>
<tr><th>ADCSTATUS</th><th>Description</th></tr>
<tr><td>0b000</td><td>IDLE or Offset cancel/refresh</td></tr>
<tr><td>0b001</td><td>Power Down</td></tr>
<tr><td>0b010</td><td>Wait State</td></tr>
<tr><td>0b100</td><td>Sample</td></tr>
<tr><td>0b110</td><td>Conversion</td></tr>
</table>
</td></tr>
</table>

34.3.2.3 Interrupt Status Register (ISR)

The ISR register contains interrupt status bits for the ADC.

Address: ADC_BASE + 0x0010 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	OFF CANC OVR	EOFF SET	EO CTU	JEOC	JECH	EOC	ECH
W										w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 34-4. Interrupt Status Register (ISR)

Table 34-4. ISR Field Descriptions

Field	Description
OFFCANCOVR	Offset Cancellation Phase Over interrupt (OFFCANCOVR) flag. When the ADC generates the offset_ok_i to high indication then the offset cancellation phase programmed by the user is over and the offset coefficient is written into the offset register. When this bit is set, an OFFCANCOVR interrupt has occurred.
EOFFSET	Error in Offset Refresh interrupt (EOFFSET) flag. This interrupt is generated during the offset cancellation phase in case the offset_measure_ok_i pulse is not received. When this bit is set, an EOFFSET interrupt has occurred.
EOCTU	End of CTU Conversion interrupt (EOCTU) flag. It is the interrupt of the digital end of conversion for the CTU channel; active when set. When this bit is set, an EOCTU interrupt has occurred.
JEOC	End of Injected Channel Conversion interrupt (JEOC) flag. It is the interrupt of the digital end of conversion for the injected channel; active when set. When this bit is set, a JEOC interrupt has occurred.
JECH	End of Injected Chain Conversion interrupt (JECH) flag. It is the interrupt of the digital end of chain conversion for the injected channel; active when set. When this bit is set, a JECH interrupt has occurred.
EOC	End of Channel Conversion interrupt (EOC) flag. It is the interrupt of the digital end of conversion. When this bit is set, an EOC interrupt has occurred.
ECH	End of Chain Conversion interrupt (ECH) flag. It is the interrupt of the digital end of chain conversion. When this bit is set, an ECH interrupt has occurred.

34.3.2.4 Channel Pending Register 0 (CEOCFR0)

Three Channel Interrupt Pending Registers are provided to signal which of the 96 channels' measures have been completed. These are COCFR[0:2]. CEOCFR0 is the End of Conversion Pending Interrupt register for group 0 channels (channels 0–31).

Address: ADC_BASE + 0x0014 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	EOC CH31	EOC CH30	EOC CH29	EOC CH28	EOC CH27	EOC CH26	EOC CH25	EOC CH24	EOC CH23	EOC CH22	EOC CH21	EOC CH20	EOC CH19	EOC CH18	EOC CH17	EOC CH16
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	EOC CH15	EOC CH14	EOC CH13	EOC CH12	EOC CH11	EOC CH10	EOC CH9	EOC CH8	EOC CH7	EOC CH6	EOC CH5	EOC CH4	EOC CH3	EOC CH2	EOC CH1	EOC CH0
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 34-5. Channel Pending Registers 0 (CEOCFR0)

Table 34-5. CEOCFR0 Field Descriptions

Field	Description
EOCCH*n*	When set, the conversion of channel *n* has been completed.

34.3.2.5 Channel Pending Register 1 (CEOCFR1)

CEOCFR1 is the End of Conversion Pending Interrupt register for group 1 channels (channels 32–63).

Address: ADC_BASE + 0x0018 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	EOC CH63	EOC CH62	EOC CH61	EOC CH60	EOC CH59	EOC CH58	EOC CH57	EOC CH56	EOC CH55	EOC CH54	EOC CH53	EOC CH52	EOC CH51	EOC CH50	EOC CH49	EOC CH48
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	EOC CH47	EOC CH46	EOC CH43	EOC CH44	EOC CH43	EOC CH42	EOC CH41	EOC CH40	EOC CH39	EOC CH38	EOC CH37	EOC CH36	EOC CH35	EOC CH34	EOC CH33	EOC CH32
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 34-6. Channel Pending Register 1 (CEOCFR1)

Table 34-6. CEOCFR1 Field Descriptions

Field	Description
EOCCH*n*	When set, the conversion of channel *n* has been completed.

34.3.2.6 Channel Pending Register 2 (CEOCFR2)

CEOCFR2 is the End of Conversion Pending Interrupt register for group 2 channels (channels 64–95).

Address: ADC_BASE + 0x001C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	EOC CH95	EOC CH94	EOC CH93	EOC CH92	EOC CH91	EOC CH90	EOC CH89	EOC CH88	EOC CH87	EOC CH86	EOC CH85	EOC CH84	EOC CH83	EOC CH82	EOC CH81	EOC CH80
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	EOC CH79	EOC CH78	EOC CH77	EOC CH76	EOC CH75	EOC CH74	EOC CH73	EOC CH72	EOC CH71	EOC CH70	EOC CH69	EOC CH68	EOC CH67	EOC CH66	EOC CH65	EOC CH64
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 34-7. Channel Pending Register 2 (CEOCFR2)

Table 34-7. CEOCFR2 Field Descriptions

Field	Description
EOCCH*n*	When set, the conversion of channel *n* has been completed.

34.3.2.7 Interrupt Mask Register (IMR)

The IMR register contains the interrupt enable bits for the ADC.

Address: ADC_BASE + 0x0020 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
W																
Reset	1	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	MSK OFF CANC OVR	MSKE OFF SET	MSK EOCTU	MSK JEOC	MSK JECH	MSK EOC	MSK ECH
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 34-8. Interrupt Mask Register (IMR)

Table 34-8. IMR Field Descriptions

Field	Description
MSKOFFCANCOVR	Mask bit for Offset Cancellation Phase Over interrupt (OFFCANCOVR). When set, the OFFCANCOVR interrupt is enabled.
MSKEOFFSET	Mask bit for Error in Offset Refresh interrupt (EOFFSET). When set, the EOFFSET interrupt is enabled.

Table 34-8. IMR Field Descriptions (continued)

Field	Description
MSKEOCTU	Mask bit for End of CTU Conversion interrupt (EOCTU). When set, the EOCTU interrupt is enabled.
MSKJEOC	Mask bit for End of Injected Channel Conversion interrupt (JEOC). When set, the JEOC interrupt is enabled.
MSKJECH	Mask bit for End of Injected Chain Conversion interrupt (JECH). When set, the JECH interrupt is enabled.
MSKEOC	Mask bit for End of Channel Conversion interrupt (EOC). When set, the EOC interrupt is enabled.
MSKECH	Mask bit for End of Chain Conversion interrupt (ECH). When set, the ECH interrupt is enabled.

34.3.2.8 Channel Interrupt Mask Register 0 (CIMR0)

The CIMR0 register contains the Channel Interrupt Enable bits for group 0 channels (channels 0–31).

Address: ADC_BASE + 0x0024 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	CIM 31	CIM 30	CIM 29	CIM 28	CIM 27	CIM 26	CIM 25	CIM 24	CIM 23	CIM 22	CIM 21	CIM 20	CIM 19	CIM 18	CIM 17	CIM 16
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	CIM 15	CIM 14	CIM 13	CIM 12	CIM 11	CIM 10	CIM 9	CIM 8	CIM 7	CIM 6	CIM 5	CIM 4	CIM 3	CIM 2	CIM 1	CIM 0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 34-9. Channel Pending Register 0 (CIMR0)

Table 34-9. CIMR0 Field Descriptions

Field	Description
CIM*n*	When set, the interrupt for channel *n* is enabled.

34.3.2.9 Channel Interrupt Mask Register 1 (CIMR1)

The CIMR1 register contains the Channel Interrupt Enable bits for group 1 channels (channels 32–63).

Address: ADC_BASE + 0x0028 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	CIM 63	CIM 62	CIM 61	CIM 60	CIM 59	CIM 58	CIM 57	CIM 56	CIM 55	CIM 54	CIM 53	CIM 52	CIM 51	CIM 50	CIM 49	CIM 48
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	CIM 47	CIM 46	CIM 43	CIM 44	CIM 43	CIM 42	CIM 41	CIM 40	CIM 39	CIM 38	CIM 37	CIM 36	CIM 35	CIM 34	CIM 33	CIM 32
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 34-10. Channel Interrupt Mask Register 1 (CIMR1)

Table 34-10. CIMR1 Field Descriptions

Field	Description
CIM*n*	When set, the interrupt for channel *n* is enabled.

34.3.2.10 Channel Interrupt Mask Register 2 (CIMR2)

The CIMR2 register contains the Channel Interrupt Enable bits for group 2 channels (channels 64–95).

Address: ADC_BASE + 0x002C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	CIM 95	CIM 94	CIM 93	CIM 92	CIM 91	CIM 90	CIM 89	CIM 88	CIM 87	CIM 86	CIM 85	CIM 84	CIM 83	CIM 82	CIM 81	CIM 80
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	CIM 79	CIM 78	CIM 77	CIM 76	CIM 75	CIM 74	CIM 73	CIM 72	CIM 71	CIM 70	CIM 69	CIM 68	CIM 67	CIM 66	CIM 65	CIM 64
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 34-11. Channel Interrupt Mask Register 2 (CIMR2)

Table 34-11. CIMR2 Field Descriptions

Field	Description
CIM*n*	When set, the interrupt for channel *n* is enabled.

34.3.2.11 Watchdog Threshold Interrupt Status Register (WTISR)

The WTISR register contains status bits for the programmable analog watchdog timer.

Address: ADC_BASE + 0x0030 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	WDG 3H	WDG 2H	WDG 1H	WDG 0H	WDG 3L	WDG 2L	WDG 1L	WDG 0L
W									w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 34-12. Watchdog Threshold Interrupt Status Register (WTISR)

Table 34-12. WTISR Field Descriptions

Field	Description
WDG*n*H	This corresponds to the status flag generated when the converted value is higher than the programmed higher threshold.
WDG*n*L	This corresponds to the status flag generated when the converted value is lower than the programmed lower threshold.

34.3.2.12 Watchdog Threshold Interrupt Mask Register (WTIMR)

The WTIMR register contains the threshold interrupt enable bits.

Address: ADC_BASE + 0x0030 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	MSK WDG 3H	MSK WDG 2H	MSK WDG 1H	MSK WDG 0H	MSK WDG 3L	MSK WDG 2L	MSK WDG 1L	MSK WDG 0L
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 34-13. Watchdog Threshold Interrupt Mask Register (WTIMR)

Table 34-13. WTISR Field Descriptions

Field	Description
MSKWDG*n*H	This corresponds to the mask bit for the interrupt generated when the converted value is higher than the programmed higher threshold. When set, the interrupt is enabled.
MSKWDG*n*L	This corresponds to the mask bit for the interrupt generated when the converted value is lower than the programmed lower threshold. When set, the interrupt is enabled.

34.3.2.13 DMA Enable Register (DMAE)

The DMAE register sets up the DMA for use with the ADC.

Address: ADC_BASE + 0x0040 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	DCLR	DMA EN
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 34-14. DMA Enable Register (DMAE)

Table 34-14. DMAE Field Descriptions

Field	Description
DCLR	DMA Clear sequence enable. 0 DMA request cleared by Acknowledge from DMA controller. 1 DMA request cleared on read of data registers.
DMAEN	DMA global enable. 0 DMA feature is disabled. 1 DMA feature is enabled.

NOTE

The ADC_DMAE[DCLR] should not be used. When ADC_DMEAE[DCLR] is set the DMA request should be cleared only after the data registers are read. However, the DMA request is automatically cleared and will not be recognised by the eDMA.

34.3.2.14 DMA Channel Select Register 0 (DMAR0)

The DMAR0 register contains the DMA Enable bits for group 0 channels (channels 0–31).

Address: ADC_BASE + 0x0044 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	DMA 31	DMA 30	DMA 29	DMA 28	DMA 27	DMA 26	DMA 25	DMA 24	DMA 23	DMA 22	DMA 21	DMA 20	DMA 19	DMA 18	DMA 17	DMA 16
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	DMA 15	DMA 14	DMA 13	DMA 12	DMA 11	DMA 10	DMA 9	DMA 8	DMA 7	DMA 6	DMA 5	DMA 4	DMA 3	DMA 2	DMA 1	DMA 0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 34-15. DMA Channel Select Register 0 (DMAR0)

Table 34-15. DMAR0 Field Descriptions

Field	Description
DMA*n*	When set, channel *n* is enabled to transfer data in DMA mode.

34.3.2.15 DMA Channel Select Register 1 (DMAR1)

The DMAR1 register contains the DMA Enable bits for group 1 channels (channels 32–63).

Address: ADC_BASE + 0x0048 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	DMA 63	DMA 62	DMA 61	DMA 60	DMA 59	DMA 58	DMA 57	DMA 56	DMA 55	DMA 54	DMA 53	DMA 52	DMA 51	DMA 50	DMA 49	DMA 48
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	DMA 47	DMA 46	DMA 43	DMA 44	DMA 43	DMA 42	DMA 41	DMA 40	DMA 39	DMA 38	DMA 37	DMA 36	DMA 35	DMA 34	DMA 33	DMA 32
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 34-16. DMA Channel Select Register 1 (DMAR1)

Table 34-16. DMAR1 Field Descriptions

Field	Description
DMA*n*	When set, channel *n* is enabled to transfer data in DMA mode.

34.3.2.16 DMA Channel Select Register 2 (DMAR2)

The DMAR2 register contains the DMA Enable bits for group 2 channels (channels 64–95).

Address: ADC_BASE + 0x004C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	DMA 95	DMA 94	DMA 93	DMA 92	DMA 91	DMA 90	DMA 89	DMA 88	DMA 87	DMA 86	DMA 85	DMA 84	DMA 83	DMA 82	DMA 81	DMA 80
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	DMA 79	DMA 78	DMA 77	DMA 76	DMA 75	DMA 74	DMA 73	DMA 72	DMA 71	DMA 70	DMA 69	DMA 68	DMA 67	DMA 66	DMA 65	DMA 64
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 34-17. DMA Channel Select Register 2 (DMAR2)

Table 34-17. DMAR2 Field Descriptions

Field	Description
DMA*n*	When set, channel *n* is enabled to transfer data in DMA mode.

34.3.2.17 Threshold Control Registers 0 – 3 (TRC*n*)

The four TRC*n* registers are used to store the user programmable upper thresholds' values.

Address: ADC_BASE + 0x0050 (TRC0)
ADC_BASE + 0x0054 (TRC1)
ADC_BASE + 0x0058 (TRC2)
ADC_BASE + 0x005C (TRC3) Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	THR EN	THR INV	THR OP	0	0	0	0	0	0	THRCH						
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 34-18. Threshold Control Registers 0 – 3 (TRC*n*)

Table 34-18. TRC*n* Field Descriptions

Field	Description
THREN	Threshold enable. When set, it enables the threshold detection feature for the selected channel.
THRINV	Invert the output pin. Setting this bit inverts the behavior of the threshold output pin.

Table 34-18. TRC*n* Field Descriptions

Field	Description
THROP	Threshold output pin status. This bit reflects the output pin status. See Section 34.4.5.1, Analog Watchdog Pulse Width Modulation Bus.
THRCH	Choose the channel (0 to 95) for threshold comparison.

34.3.2.18 Threshold Registers 0 – 3 (THRHLR*n*)

The four THRHLR*n* registers are used to store the user-programmable thresholds' values.

Address: ADC_BASE + 0x0060 (THRHLR0)
ADC_BASE + 0x0064 (THRHLR1)
ADC_BASE + 0x0068 (THRHLR2)
ADC_BASE + 0x006C (THRHLR3)

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	THRH									
W																
Reset	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	THRL									
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 34-19. Threshold Registers 0 – 3 (THRHLR*n*)

Table 34-19. THRHLR*n* Field Descriptions

Field	Description
THRH	High threshold value for channel *n*.
THRL	Low threshold value for channel *n*.

34.3.2.19 Presampling Control Register (PSCR)

Address: ADC_BASE + 0x0080 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	PREVAL2		PREVAL1		PREVAL0		PRE CONV
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 34-20. Presampling Control Register (PSCR)

Table 34-20. PSCR Field Descriptions

Field	Description
PREVAL2	Internal voltage selection for Presampling. Selects analog input voltage for presampling from the available four internal voltages (group 2 channels).
PREVAL1	Internal voltage selection for Presampling. Selects analog input voltage for presampling from the available four internal voltages (group 1 channels).
PREVAL0	Internal voltage selection for Presampling. Selects analog input voltage for presampling from the available four internal voltages (group 0 channels).
PRECONV	Convert Presampled value. If bit PRECONV is set, presampling is followed by the conversion. Sampling is bypassed and conversion of presampled data is performed.

34.3.2.20 Presampling Register 0 (PSR0)

The PSR0 register contains the Presampling Enable bits for group 0 channels (channels 0–31).

Address: ADC_BASE + 0x0084 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	PRES 31	PRES 30	PRES 29	PRES 28	PRES 27	PRES 26	PRES 25	PRES 24	PRES 23	PRES 22	PRES 21	PRES 20	PRES 19	PRES 18	PRES 17	PRES 16
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	PRES 15	PRES 14	PRES 13	PRES 12	PRES 11	PRES 10	PRES 9	PRES 8	PRES 7	PRES 6	PRES 5	PRES 4	PRES 3	PRES 2	PRES 1	PRES 0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 34-21. Presampling Register 0 (PSR0)

Table 34-21. PSR0 Field Descriptions

Field	Description
PRES*n*	When set, presampling is enabled for channel *n*.

34.3.2.21 Presampling Register 1 (PSR1)

The PSR1 register contains the Presampling Enable bits for group 1 channels (channels 32–63).

Address: ADC_BASE + 0x0088 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	PRES 63	PRES 62	PRES 61	PRES 60	PRES 59	PRES 58	PRES 57	PRES 56	PRES 55	PRES 54	PRES 53	PRES 52	PRES 51	PRES 50	PRES 49	PRES 48
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	PRES 47	PRES 46	PRES 43	PRES 44	PRES 43	PRES 42	PRES 41	PRES 40	PRES 39	PRES 38	PRES 37	PRES 36	PRES 35	PRES 34	PRES 33	PRES 32
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 34-22. Presampling Register 1 (PSR1)

Table 34-22. PSR1 Field Descriptions

Field	Description
PRES*n*	When set, presampling is enabled for channel *n*.

34.3.2.22 Presampling Register 2 (PSR2)

The PSR2 register contains the Presampling Enable bits for group 2 channels (channels 64–95).

Address: ADC_BASE + 0x008C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	PRES 95	PRES 94	PRES 93	PRES 92	PRES 91	PRES 90	PRES 89	PRES 88	PRES 87	PRES 86	PRES 85	PRES 84	PRES 83	PRES 82	PRES 81	PRES 80
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	PRES 79	PRES 78	PRES 77	PRES 76	PRES 75	PRES 74	PRES 73	PRES 72	PRES 71	PRES 70	PRES 69	PRES 68	PRES 67	PRES 66	PRES 65	PRES 64
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 34-23. Presampling Register 2 (PSR2)

Table 34-23. PSR2 Field Descriptions

Field	Description
PRES*n*	When set, presampling is enabled for channel *n*.

34.3.2.23 Conversion Timing Register 0 (CTR0)

Conversion Timing Register 0 (CTR0) is associated with group 0 channels (channels 0–31). Parameters contained in CTR0 are also used to perform Offset Cancellation and Offset Refresh.

Address: Base + 0x0094 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	INP LATCH	0	OFFSHIFT		0	INPCMP		0	INPSAMP							
W																
Reset	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	1

Figure 34-24. Conversion Timing Register 0 (CTR0)

Table 34-24. CTR0 Field Descriptions

Field	Description
INPLATCH	Configuration bit for Latching phase duration. 0 Latching phase duration is one-half clock cycle. 1 Latching phase duration is one clock cycle. **Note:** INPLATCH can be set only if INPCMP is bigger than 01b. Otherwise, INPLATCH is automatically set to 0 inside the ADC.
OFFSHIFT	Configuration bits for the Offset Shift characteristic. 00 No shift (that is the transition between codes 000h and 001h) is reached when the Avin is equal to 1 LSB. 01 Transition between code 000h and 001h is reached when the Avin is equal to 1/2 LSB. 10 Transition between code 00h and 001h is reached when the Avin is equal to 0. 11 Reserved.
INPCMP	Configuration bits for the comparison duration. See Table 34-27.
INPSAMP	Configuration bits for the sampling phase duration. See Table 34-28.

34.3.2.24 Conversion Timing Register 1 (CTR1)

Conversion Timing Register 1 (CTR1) is associated with group 1 channels (channels 32–63).

Address: ADC_BASE + 0x0098 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	INP LATCH	0	0	0	0	INPCMP		0	INPSAMP							
W																
Reset	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	1

Figure 34-25. Conversion Timing Register 1 (CTR1)

Table 34-25. CTR1 Field Descriptions

Field	Description
INPLATCH	Configuration bit for Latching phase duration. 0b Latching phase duration is one-half clock cycle. 1b Latching phase duration is one clock cycle. **Note:** The 1b condition is possible only if INPCMP is bigger than 01b. Otherwise, it is automatically set to 0b inside the ADC.
INPCMP	Configuration bits for the comparison duration.See Table 34-27.
INPSAMP	Configuration bits for the sampling phase duration. See Table 34-28.

34.3.2.25 Conversion Timing Register 2 (CTR2)

Conversion Timing Register 2 (CTR2) is associated with group 2 channels (channels 64–95).

Address: ADC_BASE + 0x009C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	INP LATCH	0	w	0	0	INPCMP		0	INPSAMP							
W																
Reset	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	1

Figure 34-26. Conversion Timing Register 2 (CTR2)

Table 34-26. CTR2 Field Descriptions

Field	Description
INPLATCH	Configuration bit for Latching phase duration. 0b Latching phase duration is one-half clock cycle. 1b Latching phase duration is one clock cycle. **Note:** The 1b condition is possible only if INPCMP is bigger than 01b. Otherwise, it is automatically set to 0b inside the ADC.
INPCMP	Configuration bits for the comparison duration. See Table 34-27.
INPSAMP	Configuration bits for the sampling phase duration. See Table 34-28.

Table 34-27. Max AD_clk Frequency and Related Configuration Settings

INPLATCH	INPCMP	INPSAMPLE	AD_clk f_{max} (MHz)	AD_clk Phase Min Duration		T_{sample} min (ns)
				High	Low	
0	0b00 or 0b01	0b0000 0011	20	24	22.5	125
0	0b00 or 0b01	0b0000 0100	20 + 4%	24	22.5	168
1	0b10	0b0000 0100	20 + 4%	24	15	168
1	0b10	0b0000 0101	20 + 4%	15	15	135
1	0b11	0b0000 0110	32 + 4%	12	12	132
1	0b11	0b0000 0111	40 + 4%	9	9	128
1	0b11	0b0000 1000	50 + 4%	9	9	134
1	0b11	0b0000 1001	60 + 4%	7.5	7.5	128

Table 34-28. ADC Sampling and Conversion Timing

INPLATCH	INPCMP	INPSAMP	T_{sample}[1]	T_{eval}	ndelay	T_{conv}
1	0b11	0b0000 1001	8 * Tck	30 * Tck	1 * Tck	39 * Tck[2]
1	0b11	0b0000 1010	9 * Tck	30 * Tck	1 * Tck	40 * Tck
1	0b11	0b0000 1011	10 * Tck	30 * Tck	1 * Tck	41 * Tck
1	0b11	0b0000 1100	11 * Tck	30 * Tck	1 * Tck	42 * Tck
1	0b11	0b0000 1101	12 * Tck	30 * Tck	1 * Tck	43 * Tck
1	0b11	0b0000 1110	13 * Tck	30 * Tck	1 * Tck	44 * Tck
1	0b11	0b0000 1111	14 * Tck	30 * Tck	1 * Tck	45 * Tck
1	0b11	0b0001 0000	15 * Tck	30 * Tck	1 * Tck	46 * Tck
1	0b11	0b0001 0001	16 * Tck	30 * Tck	1 * Tck	47 * Tck
1	0b11	0b0001 0010	17 * Tck	30 * Tck	1 * Tck	48 * Tck
1	0b11	0b0001 0011	18 * Tck	30 * Tck	1 * Tck	49 * Tck
1	0b11	0b0001 0100	19 * Tck	30 * Tck	1 * Tck	50 * Tck
1	0b11	0b0001 0101	20 * Tck	30 * Tck	1 * Tck	51 * Tck
1	0b11	0b0001 0110	21 * Tck	30 * Tck	1 * Tck	52 * Tck

Table 34-28. ADC Sampling and Conversion Timing (continued)

INPLATCH	INPCMP	INPSAMP	T_{sample}[1]	T_{eval}	ndelay	T_{conv}
1	0b11	0b0001 0111	22 * Tck	30 * Tck	1 * Tck	53 * Tck
1	0b11	0b0001 1000	23 * Tck	30 * Tck	1 * Tck	54 * Tck
1	0b11	0b0001 1001	24 * Tck	30 * Tck	1 * Tck	55 * Tck
1	0b11	0b0001 1010	25 * Tck	30 * Tck	1 * Tck	56 * Tck
1	0b11	0b0001 1011	26 * Tck	30 * Tck	1 * Tck	57 * Tck
1	0b11	0b0001 1100	27 * Tck	30 * Tck	1 * Tck	58 * Tck
1	0b11	0b0001 1101	28 * Tck	30 * Tck	1 * Tck	59 * Tck
1	0b11	0b0001 1110	29 * Tck	30 * Tck	1 * Tck	60 * Tck
1	0b11	0b0001 1111	30 * Tck	30 * Tck	1 * Tck	61 * Tck
1	0b11	0b0010 0000	31 * Tck	30 * Tck	1 * Tck	62 * Tck
1	0b11	0b0010 0001	32 * Tck	30 * Tck	1 * Tck	63 * Tck
1	0b11	0b0010 0010	33 * Tck	30 * Tck	1 * Tck	64 * Tck
1	0b11	0b0010 0011	34 * Tck	30 * Tck	1 * Tck	65 * Tck
1	0b11	0b0010 0100	35 * Tck	30 * Tck	1 * Tck	66 * Tck
1	0b11	0b0010 0101	36 * Tck	30 * Tck	1 * Tck	67 * Tck
1	0b11	0b0010 0110	37 * Tck	30 * Tck	1 * Tck	68 * Tck
1	0b11	0b0010 0111	38 * Tck	30 * Tck	1 * Tck	69 * Tck
1	0b11	0b0010 1000	39 * Tck	30 * Tck	1 * Tck	70 * Tck
1	0b11	0b0010 1001	40 * Tck	30 * Tck	1 * Tck	71 * Tck
1	0b11	0b0010 1010	41 * Tck	30 * Tck	1 * Tck	72 * Tck
1	0b11	0b0010 1011	42 * Tck	30 * Tck	1 * Tck	73 * Tck
1	0b11	0b0010 1100	43 * Tck	30 * Tck	1 * Tck	74 * Tck
1	0b11	0b0010 1101	44 * Tck	30 * Tck	1 * Tck	75 * Tck
1	0b11	0b0010 1110	45 * Tck	30 * Tck	1 * Tck	76 * Tck
1	0b11	0b0010 1111	46 * Tck	30 * Tck	1 * Tck	77 * Tck
1	0b11	0b0011 0000	47 * Tck	30 * Tck	1 * Tck	78 * Tck
1	0b11	0b0011 0001	48 * Tck	30 * Tck	1 * Tck	79 * Tck
1	0b11	0b0011 0010	49 * Tck	30 * Tck	1 * Tck	80 * Tck
1	0b11	0b0011 0011	50 * Tck	30 * Tck	1 * Tck	81 * Tck
1	0b11	0b0011 0100	51 * Tck	30 * Tck	1 * Tck	82 * Tck
1	0b11	0b0011 0101	52 * Tck	30 * Tck	1 * Tck	83 * Tck
1	0b11	0b0011 0110	53 * Tck	30 * Tck	1 * Tck	84 * Tck
1	0b11	0b0011 0111	54 * Tck	30 * Tck	1 * Tck	85 * Tck
1	0b11	0b0011 1000	55 * Tck	30 * Tck	1 * Tck	86 * Tck
1	0b11	0b0011 1001	56 * Tck	30 * Tck	1 * Tck	87 * Tck

Table 34-28. ADC Sampling and Conversion Timing (continued)

INPLATCH	INPCMP	INPSAMP	T_{sample}[1]	T_{eval}	ndelay	T_{conv}
1	0b11	0b0011 1010	57 * Tck	30 * Tck	1 * Tck	88 * Tck
1	0b11	0b0011 1011	58 * Tck	30 * Tck	1 * Tck	89 * Tck
1	0b11	0b0011 1100	59 * Tck	30 * Tck	1 * Tck	90 * Tck
1	0b11	0b0011 1101	60 * Tck	30 * Tck	1 * Tck	91 * Tck
1	0b11	0b0011 1110	61 * Tck	30 * Tck	1 * Tck	92 * Tck
1	0b11	0b0011 1111	62 * Tck	30 * Tck	1 * Tck	93 * Tck
1	0b11	0b0100 0000	63 * Tck	30 * Tck	1 * Tck	94 * Tck
1	0b11	0b0100 0001	64 * Tck	30 * Tck	1 * Tck	95 * Tck
1	0b11	0b0100 0010	65 * Tck	30 * Tck	1 * Tck	96 * Tck
1	0b11	0b0100 0011	66 * Tck	30 * Tck	1 * Tck	97 * Tck
1	0b11	0b0100 0100	67 * Tck	30 * Tck	1 * Tck	98 * Tck
1	0b11	0b0100 0101	68 * Tck	30 * Tck	1 * Tck	99 * Tck
1	0b11	0b0100 0110	69 * Tck	30 * Tck	1 * Tck	100 * Tck
1	0b11	0b0100 0111	70 * Tck	30 * Tck	1 * Tck	101 * Tck
1	0b11	0b0100 1000	71 * Tck	30 * Tck	1 * Tck	102 * Tck
1	0b11	0b0100 1001	72 * Tck	30 * Tck	1 * Tck	103 * Tck
1	0b11	0b0100 1010	73 * Tck	30 * Tck	1 * Tck	104 * Tck
1	0b11	0b0100 1011	74 * Tck	30 * Tck	1 * Tck	105 * Tck
1	0b11	0b0100 1100	75 * Tck	30 * Tck	1 * Tck	106 * Tck
1	0b11	0b0100 1101	76 * Tck	30 * Tck	1 * Tck	107 * Tck
1	0b11	0b0100 1110	77 * Tck	30 * Tck	1 * Tck	108 * Tck
1	0b11	0b0100 1111	78 * Tck	30 * Tck	1 * Tck	109 * Tck
1	0b11	0b0101 0000	79 * Tck	30 * Tck	1 * Tck	110 * Tck
1	0b11	0b0101 0001	80 * Tck	30 * Tck	1 * Tck	111 * Tck
1	0b11	0b0101 0010	81 * Tck	30 * Tck	1 * Tck	112 * Tck
1	0b11	0b0101 0011	82 * Tck	30 * Tck	1 * Tck	113 * Tck
1	0b11	0b0101 0100	83 * Tck	30 * Tck	1 * Tck	114 * Tck
1	0b11	0b0101 0101	84 * Tck	30 * Tck	1 * Tck	115 * Tck
1	0b11	0b0101 0110	85 * Tck	30 * Tck	1 * Tck	116 * Tck
1	0b11	0b0101 0111	86 * Tck	30 * Tck	1 * Tck	117 * Tck
1	0b11	0b0101 1000	87 * Tck	30 * Tck	1 * Tck	118 * Tck
1	0b11	0b0101 1001	88 * Tck	30 * Tck	1 * Tck	119 * Tck
1	0b11	0b0101 1010	89 * Tck	30 * Tck	1 * Tck	120 * Tck
1	0b11	0b0101 1011	90 * Tck	30 * Tck	1 * Tck	121 * Tck
1	0b11	0b0101 1100	91 * Tck	30 * Tck	1 * Tck	122 * Tck

Table 34-28. ADC Sampling and Conversion Timing (continued)

INPLATCH	INPCMP	INPSAMP	T_{sample}[1]	T_{eval}	ndelay	T_{conv}
1	0b11	0b0101 1101	92 * Tck	30 * Tck	1 * Tck	123 * Tck
1	0b11	0b0101 1110	93 * Tck	30 * Tck	1 * Tck	124 * Tck
1	0b11	0b0101 1111	94 * Tck	30 * Tck	1 * Tck	125 * Tck
1	0b11	0b0110 0000	95 * Tck	30 * Tck	1 * Tck	126 * Tck
1	0b11	0b0110 0001	96 * Tck	30 * Tck	1 * Tck	127 * Tck
1	0b11	0b0110 0010	97 * Tck	30 * Tck	1 * Tck	128 * Tck
1	0b11	0b0110 0011	98 * Tck	30 * Tck	1 * Tck	129 * Tck
1	0b11	0b0110 0100	99 * Tck	30 * Tck	1 * Tck	130 * Tck
1	0b11	0b0110 0101	100 * Tck	30 * Tck	1 * Tck	131 * Tck
1	0b11	0b0110 0110	101 * Tck	30 * Tck	1 * Tck	132 * Tck
1	0b11	0b0110 0111	102 * Tck	30 * Tck	1 * Tck	133 * Tck
1	0b11	0b0110 1000	103 * Tck	30 * Tck	1 * Tck	134 * Tck
1	0b11	0b0110 1001	104 * Tck	30 * Tck	1 * Tck	135 * Tck
1	0b11	0b0110 1010	105 * Tck	30 * Tck	1 * Tck	136 * Tck
1	0b11	0b0110 1011	106 * Tck	30 * Tck	1 * Tck	137 * Tck
1	0b11	0b0110 1100	107 * Tck	30 * Tck	1 * Tck	138 * Tck
1	0b11	0b0110 1101	108 * Tck	30 * Tck	1 * Tck	139 * Tck
1	0b11	0b0110 1110	109 * Tck	30 * Tck	1 * Tck	140 * Tck
1	0b11	0b0110 1111	110 * Tck	30 * Tck	1 * Tck	141 * Tck
1	0b11	0b0111 0000	111 * Tck	30 * Tck	1 * Tck	142 * Tck
1	0b11	0b0111 0001	112 * Tck	30 * Tck	1 * Tck	143 * Tck
1	0b11	0b0111 0010	113 * Tck	30 * Tck	1 * Tck	144 * Tck
1	0b11	0b0111 0011	114 * Tck	30 * Tck	1 * Tck	145 * Tck
1	0b11	0b0111 0100	115 * Tck	30 * Tck	1 * Tck	146 * Tck
1	0b11	0b0111 0101	116 * Tck	30 * Tck	1 * Tck	147 * Tck
1	0b11	0b0111 0110	117 * Tck	30 * Tck	1 * Tck	148 * Tck
1	0b11	0b0111 0111	118 * Tck	30 * Tck	1 * Tck	149 * Tck
1	0b11	0b0111 1000	119 * Tck	30 * Tck	1 * Tck	150 * Tck
1	0b11	0b0111 1001	120 * Tck	30 * Tck	1 * Tck	151 * Tck
1	0b11	0b0111 1010	121 * Tck	30 * Tck	1 * Tck	152 * Tck
1	0b11	0b0111 1011	122 * Tck	30 * Tck	1 * Tck	153 * Tck
1	0b11	0b0111 1100	123 * Tck	30 * Tck	1 * Tck	154 * Tck
1	0b11	0b0111 1101	124 * Tck	30 * Tck	1 * Tck	155 * Tck
1	0b11	0b0111 1110	125 * Tck	30 * Tck	1 * Tck	156 * Tck
1	0b11	0b0111 1111	126 * Tck	30 * Tck	1 * Tck	157 * Tck

Table 34-28. ADC Sampling and Conversion Timing (continued)

INPLATCH	INPCMP	INPSAMP	T_{sample}[1]	T_{eval}	ndelay	T_{conv}
1	0b11	0b1000 0000	127 * Tck	30 * Tck	1 * Tck	158 * Tck
1	0b11	0b1000 0001	128 * Tck	30 * Tck	1 * Tck	159 * Tck
1	0b11	0b1000 0010	129 * Tck	30 * Tck	1 * Tck	160 * Tck
1	0b11	0b1000 0011	130 * Tck	30 * Tck	1 * Tck	161 * Tck
1	0b11	0b1000 0100	131 * Tck	30 * Tck	1 * Tck	162 * Tck
1	0b11	0b1000 0101	132 * Tck	30 * Tck	1 * Tck	163 * Tck
1	0b11	0b1000 0110	133 * Tck	30 * Tck	1 * Tck	164 * Tck
1	0b11	0b1000 0111	134 * Tck	30 * Tck	1 * Tck	165 * Tck
1	0b11	0b1000 1000	135 * Tck	30 * Tck	1 * Tck	166 * Tck
1	0b11	0b1000 1001	136 * Tck	30 * Tck	1 * Tck	167 * Tck
1	0b11	0b1000 1010	137 * Tck	30 * Tck	1 * Tck	168 * Tck
1	0b11	0b1000 1011	138 * Tck	30 * Tck	1 * Tck	169 * Tck
1	0b11	0b1000 1100	139 * Tck	30 * Tck	1 * Tck	170 * Tck
1	0b11	0b1000 1101	140 * Tck	30 * Tck	1 * Tck	171 * Tck
1	0b11	0b1000 1110	141 * Tck	30 * Tck	1 * Tck	172 * Tck
1	0b11	0b1000 1111	142 * Tck	30 * Tck	1 * Tck	173 * Tck
1	0b11	0b1001 0000	143 * Tck	30 * Tck	1 * Tck	174 * Tck
1	0b11	0b1001 0001	144 * Tck	30 * Tck	1 * Tck	175 * Tck
1	0b11	0b1001 0010	145 * Tck	30 * Tck	1 * Tck	176 * Tck
1	0b11	0b1001 0011	146 * Tck	30 * Tck	1 * Tck	177 * Tck
1	0b11	0b1001 0100	147 * Tck	30 * Tck	1 * Tck	178 * Tck
1	0b11	0b1001 0101	148 * Tck	30 * Tck	1 * Tck	179 * Tck
1	0b11	0b1001 0110	149 * Tck	30 * Tck	1 * Tck	180 * Tck
1	0b11	0b1001 0111	150 * Tck	30 * Tck	1 * Tck	181 * Tck
1	0b11	0b1001 1000	151 * Tck	30 * Tck	1 * Tck	182 * Tck
1	0b11	0b1001 1001	152 * Tck	30 * Tck	1 * Tck	183 * Tck
1	0b11	0b1001 1010	153 * Tck	30 * Tck	1 * Tck	184 * Tck
1	0b11	0b1001 1011	154 * Tck	30 * Tck	1 * Tck	185 * Tck
1	0b11	0b1001 1100	155 * Tck	30 * Tck	1 * Tck	186 * Tck
1	0b11	0b1001 1101	156 * Tck	30 * Tck	1 * Tck	187 * Tck
1	0b11	0b1001 1110	157 * Tck	30 * Tck	1 * Tck	188 * Tck
1	0b11	0b1001 1111	158 * Tck	30 * Tck	1 * Tck	189 * Tck
1	0b11	0b1010 0000	159 * Tck	30 * Tck	1 * Tck	190 * Tck
1	0b11	0b1010 0001	160 * Tck	30 * Tck	1 * Tck	191 * Tck
1	0b11	0b1010 0010	161 * Tck	30 * Tck	1 * Tck	192 * Tck

Table 34-28. ADC Sampling and Conversion Timing (continued)

INPLATCH	INPCMP	INPSAMP	T_{sample}[1]	T_{eval}	ndelay	T_{conv}
1	0b11	0b1010 0011	162 * Tck	30 * Tck	1 * Tck	193 * Tck
1	0b11	0b1010 0100	163 * Tck	30 * Tck	1 * Tck	194 * Tck
1	0b11	0b1010 0101	164 * Tck	30 * Tck	1 * Tck	195 * Tck
1	0b11	0b1010 0110	165 * Tck	30 * Tck	1 * Tck	196 * Tck
1	0b11	0b1010 0111	166 * Tck	30 * Tck	1 * Tck	197 * Tck
1	0b11	0b1010 1000	167 * Tck	30 * Tck	1 * Tck	198 * Tck
1	0b11	0b1010 1001	168 * Tck	30 * Tck	1 * Tck	199 * Tck
1	0b11	0b1010 1010	169 * Tck	30 * Tck	1 * Tck	200 * Tck
1	0b11	0b1010 1011	170 * Tck	30 * Tck	1 * Tck	201 * Tck
1	0b11	0b1010 1100	171 * Tck	30 * Tck	1 * Tck	202 * Tck
1	0b11	0b1010 1101	172 * Tck	30 * Tck	1 * Tck	203 * Tck
1	0b11	0b1010 1110	173 * Tck	30 * Tck	1 * Tck	204 * Tck
1	0b11	0b1010 1111	174 * Tck	30 * Tck	1 * Tck	205 * Tck
1	0b11	0b1011 0000	175 * Tck	30 * Tck	1 * Tck	206 * Tck
1	0b11	0b1011 0001	176 * Tck	30 * Tck	1 * Tck	207 * Tck
1	0b11	0b1011 0010	177 * Tck	30 * Tck	1 * Tck	208 * Tck
1	0b11	0b1011 0011	178 * Tck	30 * Tck	1 * Tck	209 * Tck
1	0b11	0b1011 0100	179 * Tck	30 * Tck	1 * Tck	210 * Tck
1	0b11	0b1011 0101	180 * Tck	30 * Tck	1 * Tck	211 * Tck
1	0b11	0b1011 0110	181 * Tck	30 * Tck	1 * Tck	212 * Tck
1	0b11	0b1011 0111	182 * Tck	30 * Tck	1 * Tck	213 * Tck
1	0b11	0b1011 1000	183 * Tck	30 * Tck	1 * Tck	214 * Tck
1	0b11	0b1011 1001	184 * Tck	30 * Tck	1 * Tck	215 * Tck
1	0b11	0b1011 1010	185 * Tck	30 * Tck	1 * Tck	216 * Tck
1	0b11	0b1011 1011	186 * Tck	30 * Tck	1 * Tck	217 * Tck
1	0b11	0b1011 1100	187 * Tck	30 * Tck	1 * Tck	218 * Tck
1	0b11	0b1011 1101	188 * Tck	30 * Tck	1 * Tck	219 * Tck
1	0b11	0b1011 1110	189 * Tck	30 * Tck	1 * Tck	220 * Tck
1	0b11	0b1011 1111	190 * Tck	30 * Tck	1 * Tck	221 * Tck
1	0b11	0b1100 0000	191 * Tck	30 * Tck	1 * Tck	222 * Tck
1	0b11	0b1100 0001	192 * Tck	30 * Tck	1 * Tck	223 * Tck
1	0b11	0b1100 0010	193 * Tck	30 * Tck	1 * Tck	224 * Tck
1	0b11	0b1100 0011	194 * Tck	30 * Tck	1 * Tck	225 * Tck
1	0b11	0b1100 0100	195 * Tck	30 * Tck	1 * Tck	226 * Tck
1	0b11	0b1100 0101	196 * Tck	30 * Tck	1 * Tck	227 * Tck

Table 34-28. ADC Sampling and Conversion Timing (continued)

INPLATCH	INPCMP	INPSAMP	T_{sample}[1]	T_{eval}	ndelay	T_{conv}
1	0b11	0b1100 0110	197 * Tck	30 * Tck	1 * Tck	228 * Tck
1	0b11	0b1100 0111	198 * Tck	30 * Tck	1 * Tck	229 * Tck
1	0b11	0b1100 1000	199 * Tck	30 * Tck	1 * Tck	230 * Tck
1	0b11	0b1100 1001	200 * Tck	30 * Tck	1 * Tck	231 * Tck
1	0b11	0b1100 1010	201 * Tck	30 * Tck	1 * Tck	232 * Tck
1	0b11	0b1100 1011	202 * Tck	30 * Tck	1 * Tck	233 * Tck
1	0b11	0b1100 1100	203 * Tck	30 * Tck	1 * Tck	234 * Tck
1	0b11	0b1100 1101	204 * Tck	30 * Tck	1 * Tck	235 * Tck
1	0b11	0b1100 1110	205 * Tck	30 * Tck	1 * Tck	236 * Tck
1	0b11	0b1100 1111	206 * Tck	30 * Tck	1 * Tck	237 * Tck
1	0b11	0b1101 0000	207 * Tck	30 * Tck	1 * Tck	238 * Tck
1	0b11	0b1101 0001	208 * Tck	30 * Tck	1 * Tck	239 * Tck
1	0b11	0b1101 0010	209 * Tck	30 * Tck	1 * Tck	240 * Tck
1	0b11	0b1101 0011	210 * Tck	30 * Tck	1 * Tck	241 * Tck
1	0b11	0b1101 0100	211 * Tck	30 * Tck	1 * Tck	242 * Tck
1	0b11	0b1101 0101	212 * Tck	30 * Tck	1 * Tck	243 * Tck
1	0b11	0b1101 0110	213 * Tck	30 * Tck	1 * Tck	244 * Tck
1	0b11	0b1101 0111	214 * Tck	30 * Tck	1 * Tck	245 * Tck
1	0b11	0b1101 1000	215 * Tck	30 * Tck	1 * Tck	246 * Tck
1	0b11	0b1101 1001	216 * Tck	30 * Tck	1 * Tck	247 * Tck
1	0b11	0b1101 1010	217 * Tck	30 * Tck	1 * Tck	248 * Tck
1	0b11	0b1101 1011	218 * Tck	30 * Tck	1 * Tck	249 * Tck
1	0b11	0b1101 1100	219 * Tck	30 * Tck	1 * Tck	250 * Tck
1	0b11	0b1101 1101	220 * Tck	30 * Tck	1 * Tck	251 * Tck
1	0b11	0b1101 1110	221 * Tck	30 * Tck	1 * Tck	252 * Tck
1	0b11	0b1101 1111	222 * Tck	30 * Tck	1 * Tck	253 * Tck
1	0b11	0b1110 0000	223 * Tck	30 * Tck	1 * Tck	254 * Tck
1	0b11	0b1110 0001	224 * Tck	30 * Tck	1 * Tck	255 * Tck
1	0b11	0b1110 0010	225 * Tck	30 * Tck	1 * Tck	256 * Tck
1	0b11	0b1110 0011	226 * Tck	30 * Tck	1 * Tck	257 * Tck
1	0b11	0b1110 0100	227 * Tck	30 * Tck	1 * Tck	258 * Tck
1	0b11	0b1110 0101	228 * Tck	30 * Tck	1 * Tck	259 * Tck
1	0b11	0b1110 0110	229 * Tck	30 * Tck	1 * Tck	260 * Tck
1	0b11	0b1110 0111	230 * Tck	30 * Tck	1 * Tck	261 * Tck
1	0b11	0b1110 1000	231 * Tck	30 * Tck	1 * Tck	262 * Tck

Table 34-28. ADC Sampling and Conversion Timing (continued)

INPLATCH	INPCMP	INPSAMP	T_{sample}[1]	T_{eval}	ndelay	T_{conv}
1	0b11	0b1110 1001	232 * Tck	30 * Tck	1 * Tck	263 * Tck
1	0b11	0b1110 1010	233 * Tck	30 * Tck	1 * Tck	264 * Tck
1	0b11	0b1110 1011	234 * Tck	30 * Tck	1 * Tck	265 * Tck
1	0b11	0b1110 1100	235 * Tck	30 * Tck	1 * Tck	266 * Tck
1	0b11	0b1110 1101	236 * Tck	30 * Tck	1 * Tck	267 * Tck
1	0b11	0b1110 1110	237 * Tck	30 * Tck	1 * Tck	268 * Tck
1	0b11	0b1110 1111	238 * Tck	30 * Tck	1 * Tck	269 * Tck
1	0b11	0b1111 0000	239 * Tck	30 * Tck	1 * Tck	270 * Tck
1	0b11	0b11110001	240 * Tck	30 * Tck	1 * Tck	271 * Tck
1	0b11	0b1111 0010	241 * Tck	30 * Tck	1 * Tck	272 * Tck
1	0b11	0b1111 0011	242 * Tck	30 * Tck	1 * Tck	273 * Tck
1	0b11	0b1111 0100	243 * Tck	30 * Tck	1 * Tck	274 * Tck
1	0b11	0b1111 0101	244 * Tck	30 * Tck	1 * Tck	275 * Tck
1	0b11	0b1111 0110	245 * Tck	30 * Tck	1 * Tck	276 * Tck
1	0b11	0b1111 0111	246 * Tck	30 * Tck	1 * Tck	277 * Tck
1	0b11	0b1111 1000	247 * Tck	30 * Tck	1 * Tck	278 * Tck
1	0b11	0b1111 1001	248 * Tck	30 * Tck	1 * Tck	279 * Tck
1	0b11	0b1111 1010	249 * Tck	30 * Tck	1 * Tck	280 * Tck
1	0b11	0b1111 1011	250 * Tck	30 * Tck	1 * Tck	281 * Tck
1	0b11	0b1111 1100	251 * Tck	30 * Tck	1 * Tck	282 * Tck
1	0b11	0b1111 1101	252 * Tck	30 * Tck	1 * Tck	283 * Tck
1	0b11	0b1111 1110	253 * Tck	30 * Tck	1 * Tck	284 * Tck
1	0b11	0b1111 1111	254 * Tck	30 * Tck	1 * Tck	285 * Tck

1 Represents the number of clock cycles that this operation will last.

2 The ADC minimum conversion time at 60 MHz is 39 * Tck; that corresponds to 650 ns.

34.3.2.26 Normal Conversion Mask Register 0 (NCMR0)

The Normal Conversion Mask Register 0 (NCMR0) is used to program which of the group 0 channels (channel 0–31) are converted during normal conversion.

Address: ADC_BASE + 0x00A4 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	CH31	CH30	CH29	CH28	CH27	CH26	CH25	CH24	CH23	CH22	CH21	CH20	CH19	CH18	CH17	CH16
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	CH15	CH14	CH13	CH12	CH11	CH10	CH9	CH8	CH7	CH6	CH5	CH4	CH3	CH2	CH1	CH0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 34-27. Normal Conversion Mask Register 0 (NCMR0)

Table 34-29. NCMR0 Field Descriptions

Field	Description
CH*n*	When set, conversion is enabled for channel *n*.

34.3.2.27 Normal Conversion Mask Register 1 (NCMR1)

The Normal Conversion Mask Register 1 (NCMR1) is used to program which of the group 1 channels (channel 32–63) are converted during normal conversion.

Address: ADC_BASE + 0x00A8 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	CH63	CH62	CH61	CH60	CH59	CH58	CH57	CH56	CH55	CH54	CH53	CH52	CH51	CH50	CH49	CH48
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	CH47	CH46	CH45	CH44	CH43	CH42	CH41	CH40	CH39	CH38	CH37	CH36	CH35	CH34	CH33	CH32
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 34-28. Normal Conversion Mask Register 1 (NCMR1)

Table 34-30. NCMR1 Field Descriptions

Field	Description
CH*n*	When set, conversion is enabled for channel *n*.

34.3.2.28 Normal Conversion Mask Register 2 (NCMR2)

The Normal Conversion Mask Register 2 (NCMR2) is used to program which of the group 2 channels (channel 63–95) are converted during normal conversion.

Address: ADC_BASE + 0x00AC Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	CH95	CH94	CH93	CH92	CH91	CH90	CH89	CH88	CH87	CH86	CH85	CH84	CH83	CH82	CH81	CH80
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	CH79	CH78	CH77	CH76	CH75	CH74	CH73	CH72	CH71	CH70	CH69	CH68	CH67	CH66	CH65	CH64
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 34-29. Normal Conversion Mask Register 2 (NCMR2)

Table 34-31. NCMR2 Field Descriptions

Field	Description
CH*n*	When set, conversion is enabled for channel *n*.

34.3.2.29 Injected Conversion Mask Register 0 (JCMR0)

The Injected Conversion Mask Register 0 (JCMR0) is used to program which of the group 0 channels (channel 0–31) are converted during injected conversion.

Address: ADC_BASE + 0x00B4 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	CH31	CH30	CH29	CH28	CH27	CH26	CH25	CH24	CH23	CH22	CH21	CH20	CH19	CH18	CH17	CH16
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	CH15	CH14	CH13	CH12	CH11	CH10	CH9	CH8	CH7	CH6	CH5	CH4	CH3	CH2	CH1	CH0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 34-30. Injected Conversion Mask Register 0 (JCMR0)

Table 34-32. JCMR0 Field Descriptions

Field	Description
CH*n*	When set, injected sampling is enabled for channel *n*.

34.3.2.30 Injected Conversion Mask Register 1 (JCMR1)

The Injected Conversion Mask Register 1 (JCMR1) is used to program which of the group 1 channels (channel 32–63) are converted during injected conversion.

Address: ADC_BASE + 0x00B8 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	CH63	CH62	CH61	CH60	CH59	CH58	CH57	CH56	CH55	CH54	CH53	CH52	CH51	CH50	CH49	CH48
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	CH47	CH46	CH45	CH44	CH43	CH42	CH41	CH40	CH39	CH38	CH37	CH36	CH35	CH34	CH33	CH32
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 34-31. Injected Conversion Mask Register 1 (JCMR1)

Table 34-33. JCMR1 Field Descriptions

Field	Description
CH*n*	When set, injected sampling is enabled for channel *n*.

34.3.2.31 Injected Conversion Mask Register 2 (JCMR2)

The Injected Conversion Mask Register 2 (JCMR2) is used to program which of the group 2 channels (channel 63–95) are converted during injected conversion.

Address: ADC_BASE + 0x00BC Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	CH95	CH94	CH93	CH92	CH91	CH90	CH89	CH88	CH87	CH86	CH85	CH84	CH83	CH82	CH81	CH80
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	CH79	CH78	CH77	CH76	CH75	CH74	CH73	CH72	CH71	CH70	CH69	CH68	CH67	CH66	CH65	CH64
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 34-32. Injected Conversion Mask Register 2 (JCMR2)

Table 34-34. JCMR2 Field Descriptions

Field	Description
CH*n*	When set, injected sampling is enabled for channel *n*.

34.3.2.32 Offset Word Register (OFFWR)

The OFFWR register allows the user to set an offset cancellation word as opposed to a calculated word.

Address: Base + 0x00EC Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	OFFSET LOAD
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	OFFSET_WORD							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 34-33. Offset Word Register (OFFWR)

Table 34-35. OFFWR Field Descriptions

Field	Description
OFFSETLOAD	Used to enable offset loading. This bit should be written before writing the OFFSET_WORD field.
OFFSET_ WORD	The offset word coefficient generated at the end of the offset cancellation phase is latched into this register. That offset word can be also written by software. In that case, it is loaded into the analog ADC and used as the offset cancellation word instead of the one calculated using the offset cancellation process. That field should be written before starting conversion.

34.3.2.33 Decode Signals Delay Register (DSDR)

The DSDR register specifies the delay between the external decode signals and the start of the sampling phase.

Address: ADC_ Base + 0x00C4 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	DSD							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 34-34. Decode Signals Delay Register (DSDR)

Table 34-36. DSDR Field Descriptions

Field	Description
DSD	The delay between the external decode signals and the start of the sampling phase. It is used to take into account the settling time of the external mux.

34.3.2.34 Power Down Exit Delay Register (PDEDR)

The PDEDR register specifies the delay between the power-down reset and the start of the next conversion.

Address: ADC_BASE + 0x00C8 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	PDED							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 34-35. Power Down Exit Delay Register (PDEDR)

Table 34-37. PDEDR Field Descriptions

Field	Description
PDED	The delay between the power down bit MCR[PWDN] reset and the start of the next conversion.

34.3.2.35 Precision Channel *n* Data Register (PRECDATAREG*n*)

The PRECDATAREG*n* registers provide conversion results for the group 0 channel (channels 0–31) data registers. Each data register gives also some information regarding the corresponding result. One PRECDATAREG*n* register is provided for each channel.

Address: See Table 34-1. Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	VALID	OVERW	RESULT	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	CDATA									
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 34-36. Precision Channel *n* Data Register (PRECDATAREG*n*)

Table 34-38. PRECDATAREG*n* Field Descriptions

Field	Description
VALID	Used to notify when the data is valid (a new value has been written). It is automatically cleared when data is read.
OVERW	Overwrite data. Used to notify when a conversion data is overwritten by a newer result.The new data is written or discarded according to the OWREN bit of MCR register.

Table 34-38. PRECDATAREG*n* Field Descriptions

Field	Description
RESULT	This bit reflects the mode of conversion for the corresponding channel. 00 Data is a result of normal conversion mode. 01 Data is a result of injected conversion mode. 10 Data is a result of CTU conversion mode. 11 reserved
CDATA	Channel 0 – 31 converted data.

34.3.2.36 Internal Channel *n* Data Register (INTDATAREG*n*)

The INTDATAREG*n* registers provide conversion results for the group 1 channel (channels 32–63) data registers. Each data register gives also some information regarding the corresponding result. One INTDATAREG*n* register is provided for each channel.

Address: See Table 34-1. Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	VALID	OVERW	RESULT	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	CDATA									
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 34-37. Internal Channel *n* Data Register (INTDATAREG*n*)

Table 34-39. INTDATAREG*n* Field Descriptions

Field	Description
VALID	Used to notify when the data is valid (a new value has been written). It is automatically cleared when data is read.
OVERW	Overwrite data. Used to notify when a conversion data is overwritten by a newer result.The new data is written or discarded according to the OWREN bit of MCR register.
RESULT	This bit reflects the mode of conversion for the corresponding channel. 00 Data is a result of normal conversion mode. 01 Data is a result of injected conversion mode. 10 Data is a result of CTU conversion mode. 11 reserved
CDATA	Channel 32 – 63 converted data.

34.3.2.37 External Channel *n* Data Register (EXTDATAREG*n*)

The EXTDATAREG*n* registers provide conversion results for the group 2 channel (channels 64–95) data registers. Each data register gives also some information regarding the corresponding result. One EXTDATAREG*n* register is provided for each channel.

Address: See Table 34-1. Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	VALID	OVERW	RESULT	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	CDATA									
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 34-38. External Channel *n* Data Register (EXTDATAREG*n*)

Table 34-40. EXTDATAREG*n* Field Descriptions

Field	Description
VALID	Used to notify when the data is valid (a new value has been written). It is automatically cleared when data is read.
OVERW	Overwrite data. Used to notify when a conversion data is overwritten by a newer result.The new data is written or discarded according to the OWREN bit of MCR register.
RESULT	This bit reflects the mode of conversion for the corresponding channel. 00 Data is a result of normal conversion mode. 01 Data is a result of injected conversion mode. 10 Data is a result of CTU conversion mode. 11 reserved
CDATA	Channel 64 – 95 converted data.

34.4 Functional Description

34.4.1 Analog Channel Conversion

Three conversion modes are available within the ADC module:

- Normal conversion
- Injected conversion
- Triggered injected conversion using CTU

34.4.1.1 Normal Conversion

Normal conversion is programmed by configuring the normal conversion mask registers (NCMR*n*) in which each channel can be individually enabled by setting the corresponding CH*n* bit. Mask registers must be programmed before starting the conversion and cannot be changed until the conversion of all the selected channels ends.

34.4.1.2 Start of Normal Conversion

A normal conversion can be started in one of two ways:

- By software (TRGEN = 0):
  - If the external trigger enable bit is reset, the conversion chain starts when the NSTART bit in MCR is set.
- By trigger (TRGEN = 1):
  - An external trigger/enable signal is detected to start the conversion. The enable pin is checked only when the conversion is started. The end and the restart of the conversion must be checked by software. This feature is enabled setting the TRGEN bit in the MCR register. Two options are available:
    - If EDGLEV (edge/level selection) bit in MCR is reset then a rising/falling edge (depending on the EDGE bit in MCR) detected in an external pin set the NSTART bit in MCR and starts the programmed conversion. EDGE = 0 means falling, EDGE = 1 means rising edge.
    - If EDGLEV (edge/level selection) bit in MCR is set, the conversion is started if and only if the NSTART bit in MCR is set and the programmed level on the external trigger pin is detected. The level is selected using the EDGE bit in MCR. EDGE = 0 means that the start of conversion is enabled if the external pin is low. If EDGE = 1, the start of conversion is enabled when the external pin is high.

The NSTART status bit of MCR is automatically set when a normal conversion starts. At the same time NSTART bit of MCR is reset, allowing the software to program a new start of conversion. In that case the new requested conversion starts after the running conversion is completed.

If the contents of all the normal conversion mask registers is zero (i.e., no channel is selected), the conversion operation is considered completed and the interrupt ECH is immediately issued after the start of conversion.

If the START bit in MCR is reset while ADC is converting, the user needs to wait for the ADC to finish the current chain conversion (i.e., ECH interrupt) before setting the START bit once again.

34.4.1.3 Normal Conversion Operating Modes

Two operating modes are available for the normal conversion:

- One Shot mode
- Scan mode

To enter one of these modes, it is necessary to program the MCR[MODE] bit. The first phase of the conversion process involves sampling the analog channel and the next phase involves the conversion phase when the sampled analog value is converted to digital as shown in Figure 34-39.

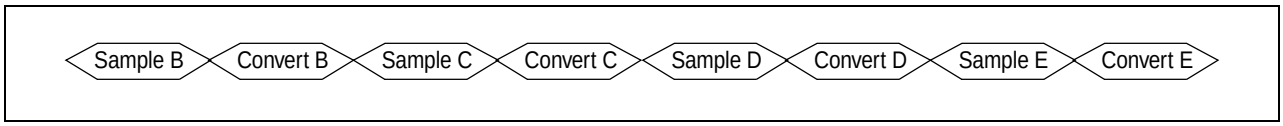


Figure 34-39. Normal Conversion Flow

In **One Shot mode** (MODE = 0), a sequential conversion specified in the NCMR*n* mask register is performed only once. At the end of each conversion, the digital result of the conversion is stored into the corresponding data register.

For example: Channels A-B-C-D-E-F-G-H are present in the device where channels B-D-E are to be converted in the One Shot mode. MODE = 0 is set for One Shot mode. Conversion starts from the channel B followed by conversion of channels D-E. At the end of conversion of channel E the scanning of channels stops.

The MSR[NSTART] status bit is automatically set when the normal conversion starts. At the same time MCR[NSTART] is reset, allowing the software to program a new start of conversion. In that case the new requested conversion starts after the running conversion is completed.

In **Scan mode** (MODE = 1), a sequential conversion of channels specified in the NCMR*n* register is continuously performed. As in the previous case, at the end of each conversion the digital result of the conversion is stored into the corresponding data register.

The MSR[NSTART] bit is automatically set when the normal conversion starts. Unlike One Shot mode, the MCR[NSTART] is not reset. It can be reset by software when the user needs to stop scan mode. In that case, the ADC completes the current scan conversion and after the last conversion resets MSR[NSTART] as well.

For Example: Channels A-B-C-D-E-F-G-H are present in the device where channels B-D-E are to be converted in the Scan mode. MODE = 1 is set for Scan mode. Conversion starts from the channel B followed by conversion of the channels D – E. At the end of conversion of channel E the scanning of channel B starts followed by conversion of the channels D – E. This sequence repeats itself until the NSTART bit of MCR is reset by software.

If the conversion is started by an external trigger and EDGLEV = 0, MCR[NSTART] bit is not set. As a consequence, once started the only way to stop scan mode conversion is to set MODE = 0.

In both modes, at the end of each conversion an End Of Conversion (EOC) interrupt is issued (if enabled by the corresponding mask bit) and at the end of the conversion sequence an End Of Chain (ECH) interrupt is issued (if enabled by the corresponding mask bit).

34.4.1.4 Injected Channel Conversion

A conversion chain can be injected into the ongoing normal conversion by configuring the JCMR*n* injected mask registers. As in normal conversion, each internal channel can be individually selected. This injected conversion can only be in one-shot mode and interrupts the normal conversion. When an injected conversion is inserted, ongoing channel conversion is aborted and the injected channel request is processed. After the last channel in the injected chain is converted, normal conversion resumes from the channel at which the normal conversion was aborted as shown in Figure 34-40.

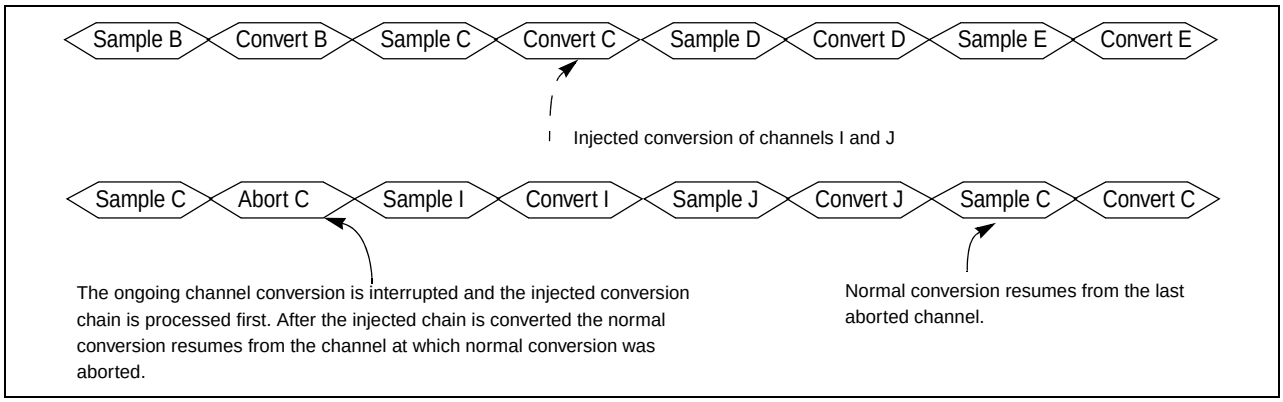


Figure 34-40. Injected Sample/Conversion Sequence

The ADC input signal “injection_trg” must be connected to the PIT_2 output. This allows the application to use PIT_2 to periodically inject conversion chains. The ADC can also be externally triggered from external inputs. See pin muxing for details.

The injected conversion can be started using two options:

- By software setting the MCR[JSTART] bit; the current conversion is suspended and the injected chain is converted. At the end of the chain, the MSR[JSTART] bit is reset and the normal chain conversion is resumed.
- By an external trigger signal setting the MCR[JTRGEN] bit; a programmed event (rising/falling edge depending on MCR[JEDGE] bit) on the injection external pin starts the injected conversion by setting the JSTART bit. At the end of the chain, the MSR[JSTART] bit is reset and the normal chain conversion is resumed.

The JSTART status bit of MSR is automatically set when the injected conversion starts. At the same time JSTART bit of MCR is reset, allowing the software to program a new start of conversion. In that case the new requested conversion starts after the running conversion is completed.

At the end of each injected conversion, an End Of Injected Conversion (JEOC) interrupt is issued (if enabled by the corresponding mask bit) and at the end of the sequence an End Of Injected Chain (JECH) interrupt is issued (if enabled by the corresponding mask bit).

If the content of all the injected conversion mask registers is zero (i.e., no channel is selected) the interrupt JECH is immediately issued after the start of conversion.

Once started, injected chain conversion cannot be interrupted.

34.4.1.5 Abort Conversion

Two different abort functions are provided.

The user can abort the ongoing conversion by setting the MCR[ABORT] bit. This results in a new start pulse to the analog ADC. In the case of an abort operation, the NSTART/JSTART bit remains set and the ABORT bit gets reset after the conversion of the next channel starts.This behavior is true for normal or triggered/injected conversion modes.

It is also possible to abort the current chain conversion setting the MCR[ABORTCHAIN] bit. In that case the behavior of the ADC depends on the MODE bit. If scan mode is disabled, the NSTART bit is automatically reset together with the ABORTCHAIN bit. Otherwise, if the MODE = 1, a new chain conversion is started.

When an ABORTCHAIN is requested while an injected conversion is running over a suspended normal conversion, both injected chain and normal conversion chain are aborted (both NSTART and JSTART bits are reset too).

34.4.2 Analog Clock Generator and Conversion Timings

The analog clock provided to the ADC module cannot be faster than 60 MHz and must have a 50% duty cycle.

As shown in Figure 34-41, the analog clock generator is made up of a clock prescaler and the AD_clk frequency is half ipg_clk frequency. When the CTU interface is enabled, depending on the position of the rising edge of the signal ctu_trigger (coming from the CTU), AD_clk could also be stretched as described in Figure 34-41.

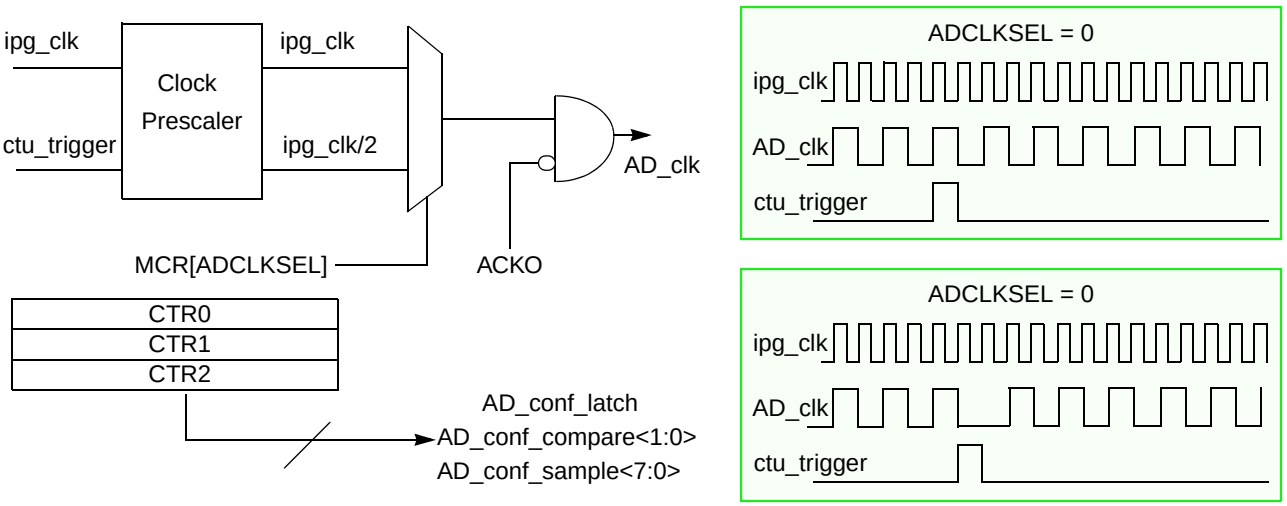


Figure 34-41. Prescaler Simplified Block Diagram

The clock stretching is implemented if and only if MCR[ADCLKSEL] = 0 (if AD_clk is half ipg_clk).

In order to support different loadings and switching times (in particular for the group 2 channels), three different Conversion Timing registers are present (CTR[0 – 2]). The first is associated with group 0 channels, the second one with group 1 channels, and the third one with the group 2 channels. INPLATCH and IPCMP configurations are limited when ipg_clk frequency is greater than 20 Mhz.

34.4.3 ADC Cross Triggering Unit

The ADC Cross Triggering Unit (CTU) is included to enhance the injected conversion capability of the ADC. The CTU contains multiple event inputs that can be used to select the channels to be converted from the appropriate event configuration register. In Figure 34-42 the CTU/ADC interface is shown. The CTU generates a trigger output pulse (*ctu_trigger*) of one clock cycle and outputs onto a data bus (*ctu_numchannel*) which channel has to be converted. A single channel is converted for each request. After performing the conversion, the ADC returns the result on the ctu_dataout bus. Together with the converted data, the ADC outputs two pulses named *ctu_nextcmd* and *ctu_push* spaced one clock cycle (of analog AD_clk) apart. The ctu_nextcmd is set to '1' during the last clock cycle of ADC evaluation phase. The ctu_push is asserted when the conversion is finished, meaning that the ctu_dataout is valid. After performing the conversion, the conversion result is saved in the corresponding data register and it is compared with Watchdog thresholds if requested.

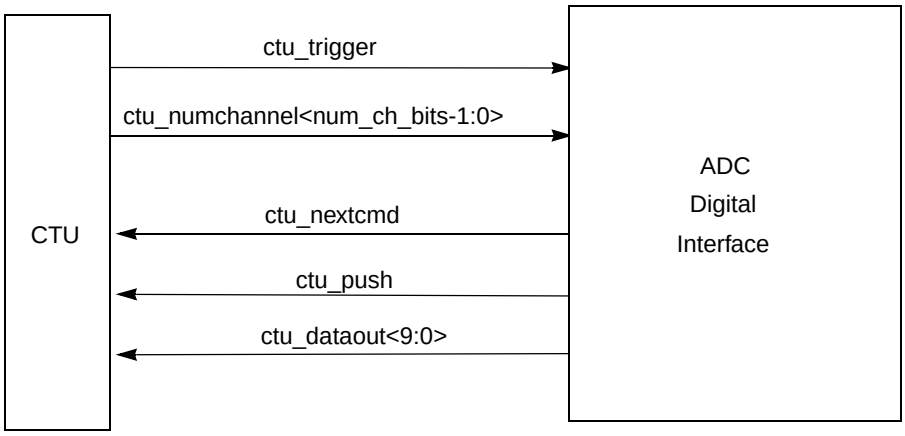


Figure 34-42. ADC Cross Triggering Unit

Two different modes can be selected by programming the CTUEN field of MCR, as described in Table 34-41:

Table 34-41. CTU Selection

CTUEN	CTU Mode
0	CTU is disabled.
1	CTU Trigger Mode is enabled.

When the CTU is enabled, the CPU is able to write in the ADC registers but the CPU cannot start any conversion. Conversion requests can be generated only by the CTU trigger pulse. If a normal or injected conversion is requested, it is automatically discarded. When the CTU sends to the ADC the number of the

channel to be converted, that is taken as the injected channel value and the triggered injected conversion starts. The CTUSTART bit is set automatically at this point and it is also automatically reset when the CTU is disabled (CTUEN = 0).

The CTU and the ADC digital interface are synchronous with the ipg_clk in both cases.

34.4.3.1 CTU Trigger Mode

In CTU trigger mode, normal and injected conversion are still enabled.

When a ctu_trigger pulse is received the output data ctu_numchannel is taken as the injected channel value and the triggered injected conversion starts. The MSR[CTUSTART] bit is set automatically at this point and it is also automatically reset when the triggered injected conversion is completed.

CTU conversions must be requested (generating ctu_trigger pulse) when offset cancellation phase is over. Otherwise, ctu_trigger pulse is discarded. If a ctu_trigger pulse is received during an offset refresh, the current refresh is immediately stopped in order to satisfy the CTU request.

If an injected conversion (programmed by the user by setting the JSTART bit) is ongoing and a pulse is received on the ctu_trigger input line, then the injected channel conversion chain is aborted) and only the triggered injected conversion proceeds. After aborting the injected conversion, the MSR[JSTART] bit is reset to ‘0’. That abort is signalled through the MSR[JABORT] status bit.

If normal conversion is ongoing and a pulse is received on the ctu_trigger input line, then ongoing channel conversion is aborted and the triggered injected conversion is processed. When it is finished, normal conversion resumes from the channel at which the normal conversion was aborted.

If another ctu_trigger pulse is received before ctu_nextcmd signal, that triggered conversion request is discarded.

When a normal conversion is requested during CTU conversion (CTUSTART = 1), the normal conversion starts when CTU conversion is completed (CTUSTART = 0). Otherwise when an injected conversion is requested during CTU conversion, that conversion is discarded and the MCR[JSTART] bit is immediately reset.

34.4.3.2 CTU Control Mode

When CTU Control mode is enabled, the CPU is able to write in the ADC registers but it cannot start any conversion. Conversion requests can be generated only by the CTU trigger pulse. If a normal or injected conversion is requested, it is automatically discarded.

When a ctu_trigger pulse is received the output data ctu_numchannel is taken as the injected channel value and the triggered injected conversion starts. The CTUSTART bit is set automatically at this point and it is also automatically reset when CTU Control mode is disabled (CTUEN = 0).

CTU conversions must be requested (generating ctu_trigger pulse) when offset cancellation phase is over. In fact, each ctu_trigger pulse received during offset cancellation is discarded. Otherwise, if a ctu_trigger pulse is received during an offset refresh, the current refresh is immediately stopped in order to satisfy the CTU request.

Timings of CTU interface signals in CTU Control mode are shown in Figure 34-43 and Figure 34-44.

In both cases, the delay between the positive edge of ctu_trigger signal and the starting of conversion is fixed. In the first case, the AD_clock is stretched during the low phase in order to guarantee that constraint.

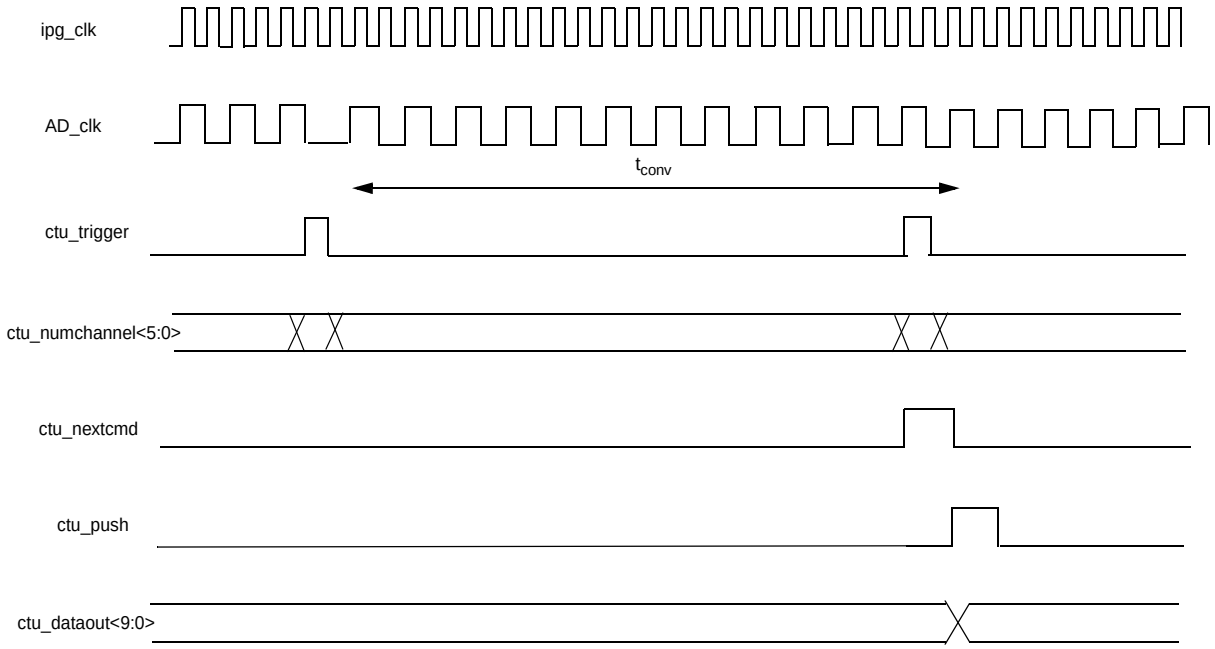


Figure 34-43. CTU Control Mode Interface Timings (Case 1)

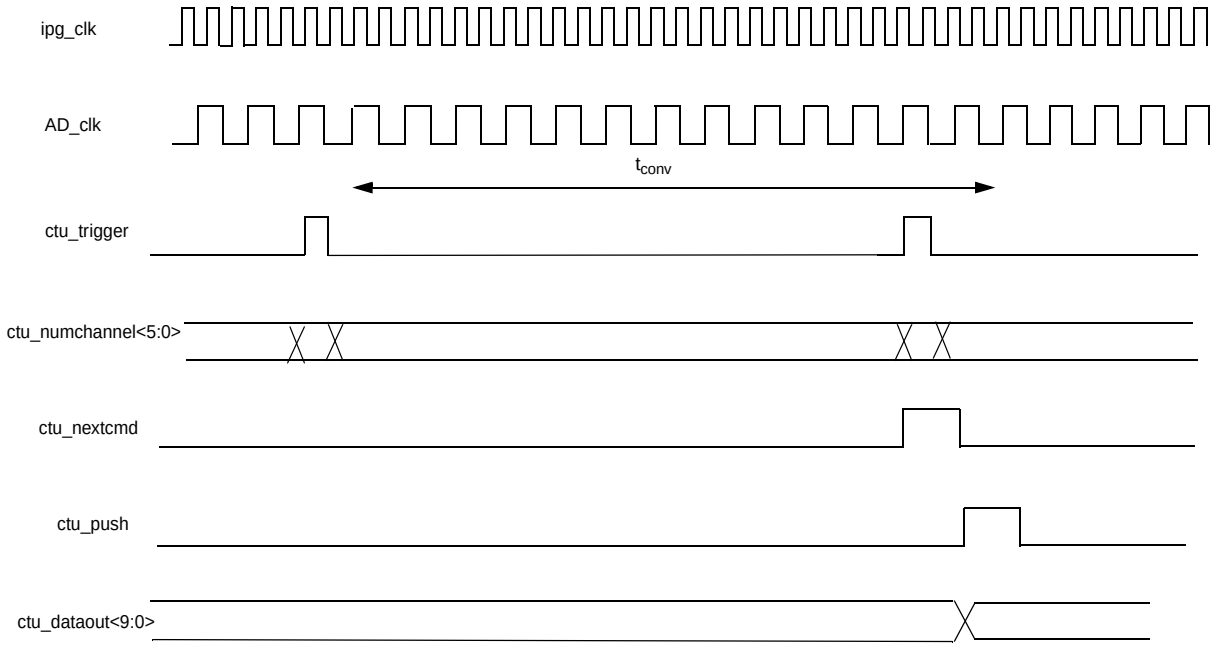


Figure 34-44. CTU Control Mode Interface Timings (Case 2)

34.4.4 Presampling

Presampling allows to precharge or discharge the ADC internal capacitor before it starts sampling/conversion of the analog input coming from pads. This is useful in the reset information (offset cancellation) regarding the last converted data. During presampling, the analog ADC samples the internally generated voltage. During sampling, the analog ADC samples analog input coming from pads.

Presampling can be enabled/disabled on a channel basis by setting the corresponding bits in the PSR*n* registers.

After enabling the presampling for a channel, the normal sequence of operation for that channel is Presampling + Sampling + Conversion. Sampling of the channel can be bypassed by setting the PRECONV bit in the PSCR register. When sampling of a channel is bypassed, the sampled data of internal voltage in the presampling state is converted.

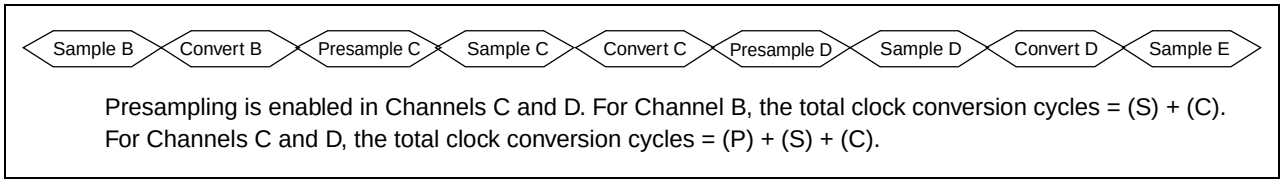


Figure 34-45. Presampling Sequence

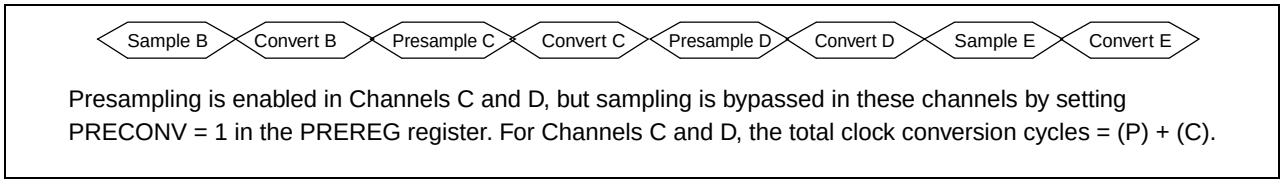


Figure 34-46. Presampling Sequence with PRECONV = 1

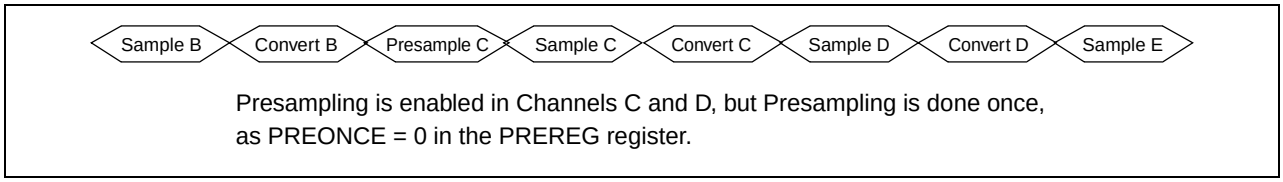


Figure 34-47. Presampling Sequence with PREONCE = 0

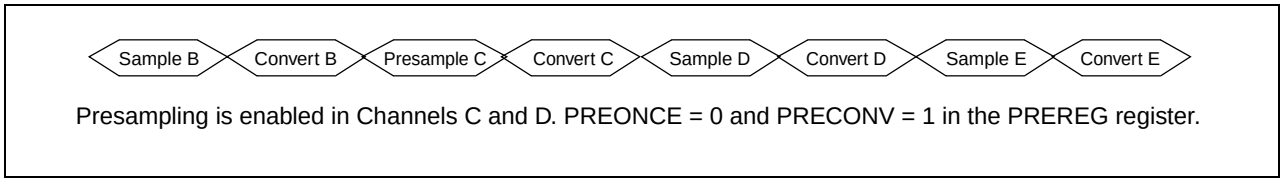


Figure 34-48. Presampling Sequence with PRECONV = 1 and PREONCE = 0

34.4.4.1 Presampling Channel Enable Signals

The presampling enable signals (presample_sw_en<3:0>) are used to enable analog switch used to sample an internally generated voltage. It is possible to select between four internally generated voltages V0,V1,V2,V3 depending on the value of PREVAL*n* fields in the PSCR register as given in Table 34-42.

Table 34-42. Presampling Voltage Selected Based on PREVAL Field

PREVAL0/PREVAL1/PREVAL2	Presampling Voltage	Voltage Signal	presample_sw_en
00	V0	V_{RL}	0b0001
01	V1	V_{RH}	ob0010
10	V2	V_{DDSYN}	0b0100
11	V3	V_{DD}	0b1000

Three Presampling Value fields (PREVAL0, PREVAL1, PREVAL2) are contained in PSCR register. The first one is associated with group 0 channels (0 to 31), the second one with group 1 channels (from 32 to 63), and the third one with group 2 channels. That allows to select three different presampling values for each channel type.

If the COUNTER bit of the PREREG is set, then it has a higher priority over the PREVAL field. In that case, the first voltage (V0) is selected in the start and used for the first chain's conversion. When the conversion of the first chain is over, the next voltage (V1) is selected for presampling of the next chain and so on.After V3 the next chain conversion presampling voltage wraps around to V0.

In case of a scan mode conversion if the COUNTER bit is set then for every wrap around of the conversion chain, the value of the presampling voltage employed to precharge the ADC hard macrocell is changed as following: For the first chain a value of V0 is used, for the second chain V1 is used, and so on. When the presampling voltage value reaches V3, then for the next conversion chain, the value used for presampling is V0 and the sequence repeats itself.

The presampling channel enable timings follow a different methodology in respect to the channel select timings (for detailed explanation refer to ADC analog specifications).

34.4.5 Programmable Analog Watchdog

The analog watchdogs are used for determining whether the value obtained on conversion of a channel lies within a given guard area (as shown in Figure 34-50). If the converted value lies outside the guard area specified by the upper and lower threshold values, then corresponding threshold violation interrupts are generated.

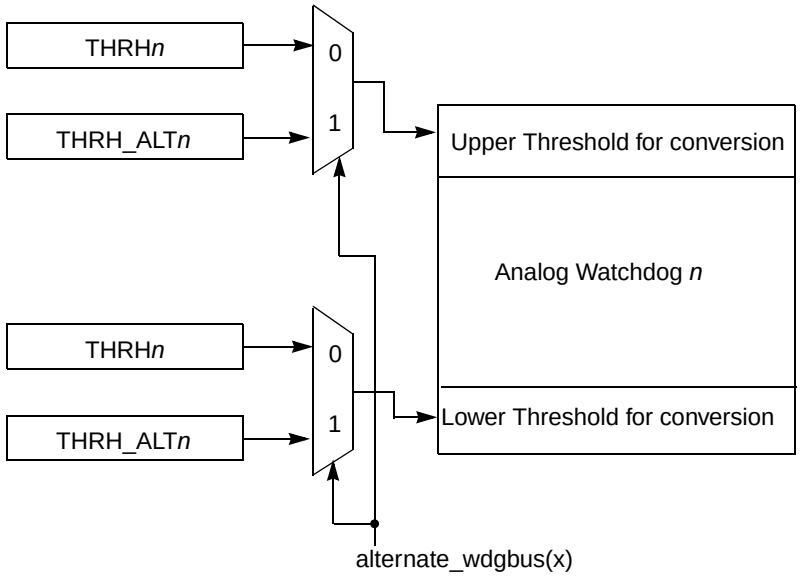
As many as four analog watchdogs are available. The channel on which the analog watchdog is to be applied is selected by the THRCH field in the TRC*n* (*n* = [0...3]) register. The analog watchdog is enabled by setting the corresponding THREN bit in TRC*n* register (*n* = [0...3]).

The lower and higher threshold values for the analog watchdog are programmed by THRL field in TRB*n* (*n* = [0...3]) and THRH field in TRA*n* (*n* = [0...3]) respectively.

The logic configuration for selecting the threshold values for the analog watchdog is shown in Figure 34-50. After the conversion of the selected channel a comparison is performed between the converted value obtained and the threshold values set. The result is stored as WDG*n*H and WDG*n*L bits in the WTISR register as explained in Table 34-43. Depending on the MSKWDG*n*L and MSKWDG*n*H mask bits in the WTIMR register, an interrupt is generated on threshold violation.

Table 34-43. Values of WDG*n*H and WDG*n*L Fields

WDG*n*H	WDG*n*L	Converted data
1	0	converted data > THRH
0	1	converted data < THRL
0	0	THRH ≤ converted data ≤ THRL



Setting Upper and Lower Threshold values for the analog watchdog [*n* = (0 ... 3)]

Figure 34-49. Alternate Watchdog Threshold Configuration

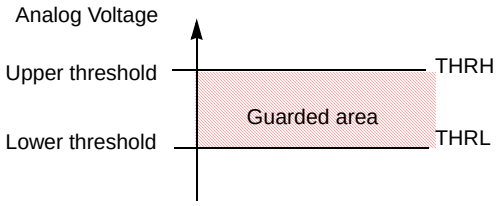


Figure 34-50. Guarded Area

NOTE

If the higher threshold for the analog watchdog is programmed lower than the lower threshold and the converted value is lesser than the lower threshold then the WDG*n*L interrupt for the low threshold violation is set, else if the converted value is greater than the lower threshold (consequently also greater than the higher threshold) then the interrupt WDG*n*H for high threshold violation is set. Thus the user should take care of avoiding that situation as it could lead to misinterpretation of the watchdog interrupts.

34.4.5.1 Analog Watchdog Pulse Width Modulation Bus

For each input channel, an output bus is used to signal outside the result of the comparison generating modulated pulse waveforms based on the converted analog values received by the analog watchdogs:

- If the converted data value is lower than the lower threshold, then the output pin (and THROP bit in TRC*n* register) is forced high.
- If the converted voltage is higher than the higher threshold, then the output pin (and THROP bit in TRC*n* register) is forced low.
- If the converted voltage lies between the upper and the lower threshold guard window, then the output pin (and THROP bit in TRC*n* register) keeps its logic value.

The logic level of the output pin can be programmed by software. In fact, the user can decide to keep the behavior described or to invert the output logic level by setting the THRINV bit in the TRC*n* register.

The values set on the ad_awpwm bus remain same in case the alternate watchdog thresholds are employed by enabling the corresponding bit of the alternate watchdog input bus.

An example of the operation is shown in Table 34-44.

Table 34-44. Example for ad_awpwm_o Operation

Converted data watchdog(*n*)	Upper Threshold watchdog(*n*)	Lower Threshold watchdog(*n*)	THRINV watchdog(*n*)	AD_AWPWM(*n*)
0x0155	0x0055	0x0000	0	0
0x0055	0x01FF	0x0088	0	1
0x0155	0x0055	0x0000	1	1
0x0055	0x01FF	0x0088	1	0

34.4.6 DMA Functionality

A Direct Memory Access (DMA) request can be programmed after the conversion of every channel by setting the respective masking bit in DMAR*n* registers. The DMA masking registers must be programmed before starting any conversion.

DMA interface signals timings are described in Figure 34-51.

DMA transfers can be enabled by setting the DMAE[DMAEN] bit. When DMAE[DCLR] is set then the DMA request is cleared on the reading of the register for which DMA transfer has been enabled.

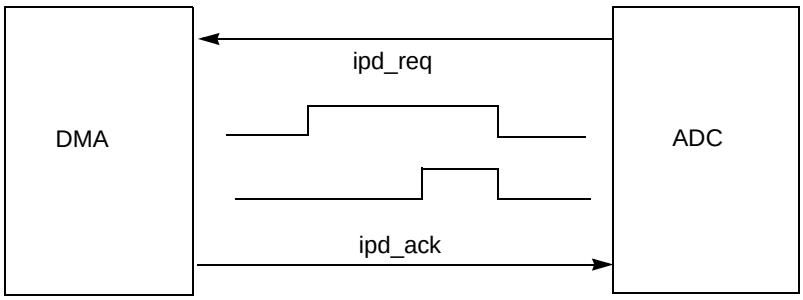


Figure 34-51. Request/Acknowledge Timings

34.4.7 Interrupts

The ADC generates the following maskable interrupt signals:

- EOC (End of Conversion) interrupt request
- ECH (End of Chain) interrupt request
- JEOC (End of Injected Conversion) interrupt request
- JECH (End of Injected Chain) interrupt request
- EOFFSET (Error in Offset Cancellation) interrupt request
- OFFCANCOVR (Offset Cancellation Phase Over) interrupt request
- WDG*n*L and WDG*n*H (Watchdog Threshold) interrupt requests

Interrupts are generated during the conversion process to signal events such as End of Conversion, etc., as explained in the register description for ISR. Two registers named ISR (Interrupt Status Register) and IMR (Interrupt Mask Register) are provided in order to check and enable the interrupt request to the External Interrupt Control (EIC) module.

The interrupts generated by the analog watchdog are handled by two registers, Watchdog Threshold Interrupt Status Register (WTISR) and Watchdog Threshold Interrupt Mask Register (WTIMR), in order to check and enable the interrupt request to EIC module. The watchdog interrupt source sets 2 pending bits WDG*n*H and WDG*n*L for each of the four channels being monitored in the WTISR register.

In order to reduce the number of interrupt lines, interrupts are combined (ORed) on three lines:

- EOC, ECH, JEOC and JECH on the ADC_EOC line
- EOFFSET and OFFCANCOVR on the ADC_ER line
- WDG0L, WDG0H, WDG1L, WDG1H, WDG2L, WDG2H, WDG3L and WDG3H on the ADC_WD line

The ISR register contains the interrupt pending request status. In case the user wants to clear a particular interrupt event status, then writing a ‘1’ to the corresponding status bit clears the pending interrupt flag (at this write operation all the other bits of the ISR register must be maintained at ‘0’). This is the only write operation on the ISR and WTISR register. Any other write operation is forbidden, so the registers are accessible in Read/Clear mode only.

34.4.8 External Decode Signals Delay

The ADC provides three external decode signals used to select which of the 32 group 2 channels has to be converted. In order to take into account the control switching time of the external analog mux a Decode Signals Delay register is provided. Writing that register allows to program the delay between the decoding signal selection and the actual start of conversion

34.4.9 Power Down Mode

The analog part of the ADC can be put in low power mode by setting the PWDN bit in MCR. After ipg_hard_async_reset_b is released the ADC analog module is kept in power down mode by default, so this state must be exited before starting any operation by resetting the appropriate bit in MCR. When in power down mode, no offset cancellation or conversion can be started. If a ctu_trigger pulse is received during power-down, it is discarded.

If a conversion or an offset cancellation is ongoing then ADC hard macrocell cannot be put immediately into the power down mode. The ongoing operation must be allowed to complete or be aborted manually (by resetting the NSTART or OFFCANC bit) and only then may the PWDN bit in the MCR register be cleared. A normal or an injected conversion can be aborted using the ABORTCHAIN function.

If the PWDN is set while a normal conversion is running, the ADC completes the current channel conversion before entering power-down. As a consequence, PWDN bit of CSR is set only after the running conversion has finished.

If the CTU is enabled and CTUSTART bit is ‘1’ then the PWDN bit cannot be set. When CTU trigger mode is enabled, the application has to wait for the end of conversion (CTUSTART bit automatically reset). When CTU control mode is enabled, before entering power-down the application needs to reset CTUEN bit, too.

After the power down phase is completed, the process ongoing before the power down phase must be restarted manually (by setting the appropriate START/OFFCANC bit).

34.4.10 Auto Clock Off Mode

To reduce the power consumption during the IDLE mode of operation (without going into power down mode), an “auto-clock-off” feature can be enabled by setting the MCR[ACKO] bit. When enabled, the analog clock is automatically switched off when no operation is ongoing, that is, no offset cancellation or offset refresh in idle phase or conversion is programmed by the user.

Chapter 35
IEEE 1149.1 Test Access Port Controller (JTAGC)

35.1 Introduction

The JTAGC provides the means to test chip functionality and connectivity and controls access to the debug features of the device while remaining transparent to system logic when not in test mode. Testing is performed via a boundary scan technique, as defined in the IEEE® 1149.1-2001 standard. Instructions can be executed that allow the test access port (TAP) to be shared with other modules on the MCU. All data input to and output from the JTAGC is communicated in serial format.

Chapter 36, Nexus Development Interface (NDI), includes information relevant to use of the JTAGC, including Section 35.6, Initialization/Application Information, which gives practical application examples for both controllers.

35.1.1 Block Diagram

A simplified block diagram of the JTAGC illustrates the functionality and interdependence of major blocks (see Figure 35-1). The JTAG port of the device consists of four inputs and one output. These pins include JTAG compliance select (JCOMP), test data input (TDI), test data output (TDO), test mode select (TMS), and test clock input (TCK). TDI, TDO, TMS, and TCK are compliant with the IEEE 1149.1-2001 standard and are shared with the NDI through the test access port (TAP) interface.

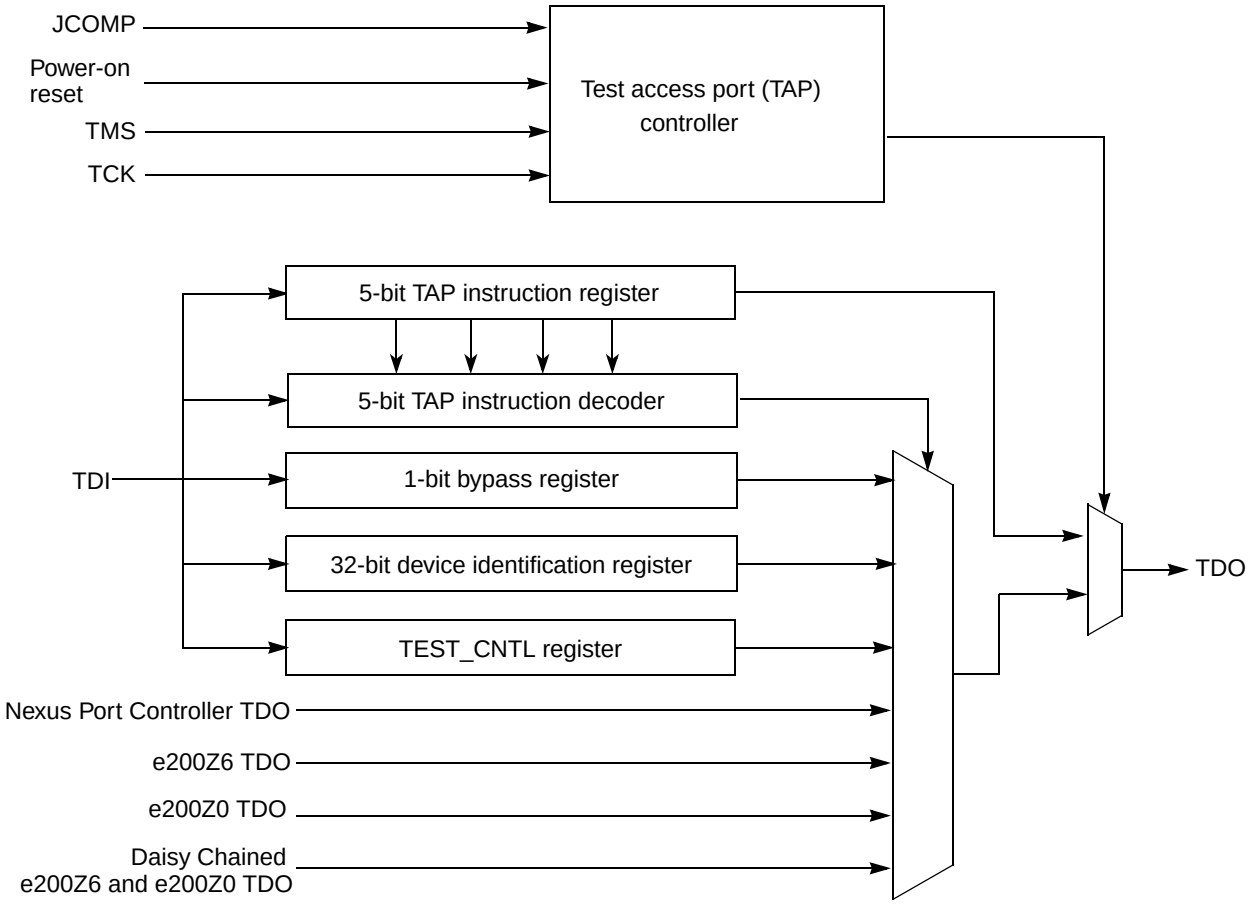


Figure 35-1. JTAGC Block Diagram

35.1.1.1 Individual and Multi-Core Debug

When daisy chaining a JTAG interface (see Figure 35-2), two separate commands can be shifted into the different cores serially. This allows start (go) commands or step commands to be input to the cores in parallel. Commands are shifted in during the JTAG SHIFT_IR state and are executed when the UPDATE_IR state is reached in the TAP state diagram.

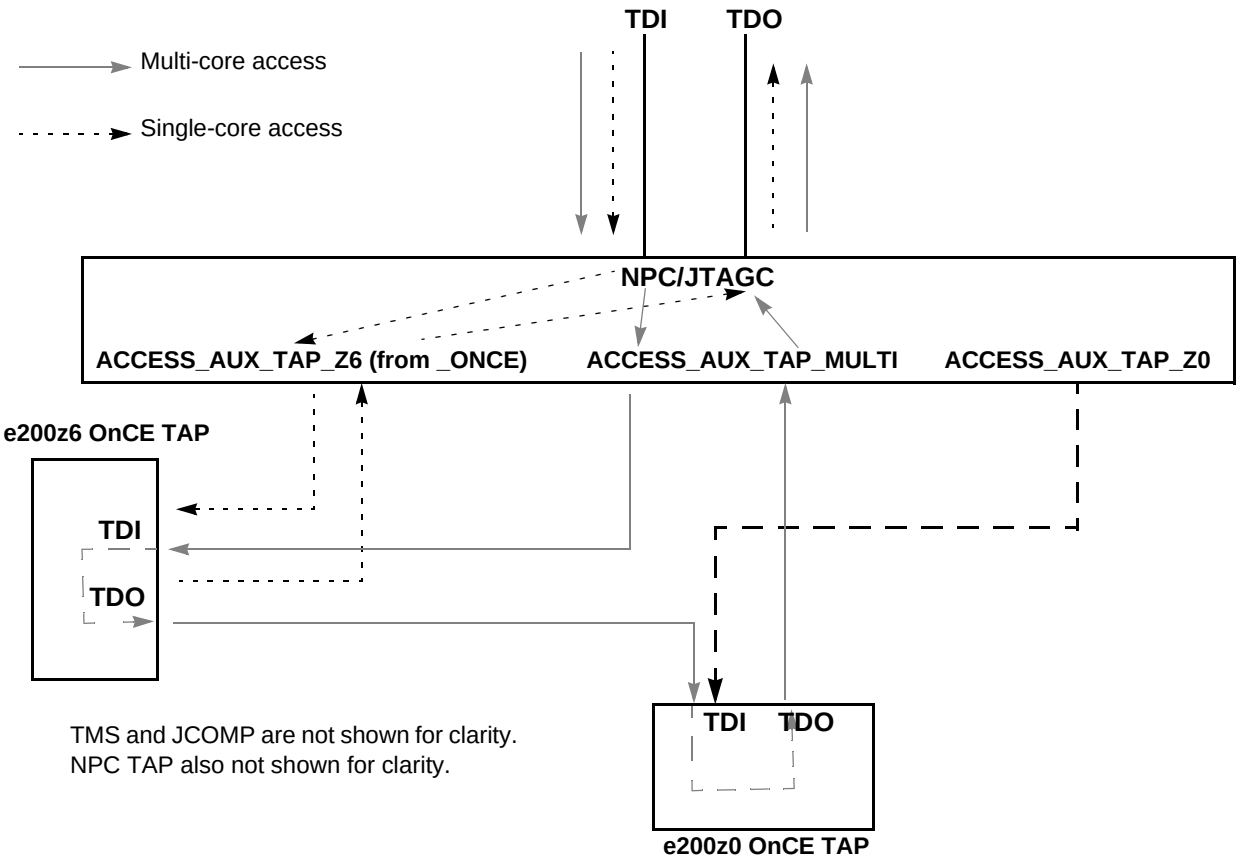


Figure 35-2. JTAG/Nexus Daisy Chain of the PXN20 e200z6 and e200z0 Cores

35.1.2 Features

The JTAGC is compliant with the IEEE 1149.1-2001 standard and has these major features:

- IEEE 1149.1-2001 test access port (TAP) interface.
- A JCOMP input that provides the ability to share the TAP.
- A 5-bit instruction register that supports several IEEE 1149.1-2001 defined instructions, as well as several public and private MCU-specific instructions (see Table 35-2).
- Three test data registers: a bypass register, a boundary scan register, and a device identification register.
- A TAP controller state machine that controls the operation of the data registers, instruction register, and associated circuitry.

35.1.3 Modes of Operation

The JTAGC uses JCOMP and a power-on reset indication as its primary reset signals. Several IEEE 1149.1-2001 defined test modes are supported, as well as a bypass mode.

35.1.3.1 Reset

The JTAGC is placed in reset when the TAP controller state machine is in the TEST-LOGIC-RESET state. The TEST-LOGIC-RESET state is entered upon the assertion of the power-on reset signal, negation of JCOMP, or through TAP controller state machine transitions controlled by TMS. Asserting power-on reset or negating JCOMP results in asynchronous entry into the reset state. While in reset, the following actions occur:

- The TAP controller is forced into the test-logic-reset state, thereby disabling the test logic and allowing normal operation of the on-chip system logic to continue unhindered.
- The instruction register is loaded with the IDCODE instruction.

In addition, execution of certain instructions can result in assertion of the internal system reset. These instructions include EXTEST, CLAMP, and HIGHZ.

35.1.3.2 IEEE 1149.1-2001 Defined Test Modes

The JTAGC supports several IEEE 1149.1-2001 defined test modes. The test mode is selected by loading the appropriate instruction into the instruction register while the JTAGC is enabled. Supported test instructions include EXTEST, HIGHZ, CLAMP, SAMPLE, and SAMPLE/PRELOAD. Each instruction defines the set of data registers that may operate and interact with the on-chip system logic while the instruction is current. Only one test data register path is enabled to shift data between TDI and TDO for each instruction.

The boundary scan register is enabled for serial access between TDI and TDO when the EXTEST, SAMPLE, or SAMPLE/PRELOAD instructions are active. The single-bit bypass register shift stage is enabled for serial access between TDI and TDO when the HIGHZ, CLAMP, or reserved instructions are active. The functionality of each test mode is explained in more detail in Section 35.4.4, JTAGC Instructions.

35.1.3.3 Bypass Mode

When no test operation is required, the BYPASS instruction can be loaded to place the JTAGC into bypass mode. While in bypass mode, the single-bit bypass shift register is used to provide a minimum-length serial path to shift data between TDI and TDO.

35.1.3.4 TAP Sharing Mode

There are three selectable auxiliary TAP controllers that share the TAP with the JTAGC. Selectable TAP controllers include the Nexus port controller (NPC), e200z6 OnCE, and e200z0. The instructions required to grant ownership of the TAP to the auxiliary TAP controllers are ACCESS_AUX_TAP_NPC, ACCESS_AUX_TAP_Z6 (from _ONCE) (for e200z6), and ACCESS_AUX_TAP_Z0. Additionally, the instruction for daisy chaining the e200z6 and e200z0 is ACCESS_AUX_TAP_MULTI (allows instructions to be clocked into both the e200z0 and e200z6 serially). Instruction opcodes for each instruction are shown in Table 35-2.

When the access instruction for an auxiliary TAP is loaded, control of the JTAG pins is transferred to the selected TAP controller. Any data input via TDI and TMS is passed to the selected TAP controller, and any

TDO output from the selected TAP controller is sent back to the JTAGC to be output on the pins. The JTAGC regains control of the JTAG port during the UPDATE-DR state if the PAUSE-DR state was entered. Auxiliary TAP controllers are held in RUN-TEST/IDLE while they are inactive.

For more information on the TAP controllers see Chapter 36, Nexus Development Interface (NDI).

35.2 External Signal Description

Refer to Table 3-1 and Section 3.4, Detailed Signal Description, for detailed signal descriptions.

35.3 Memory Map and Registers

This section provides a detailed description of the JTAGC registers accessible through the TAP interface, including data registers and the instruction register. Individual bit-level descriptions and reset states of each register are included. These registers are not memory-mapped and can be accessed through the TAP only.

35.3.1 Instruction Register

The JTAGC uses a 5-bit instruction register as shown in Figure 35-3. The instruction register allows instructions to be loaded into the module to select the test to be performed or the test data register to be accessed or both. Instructions are shifted in through TDI while the TAP controller is in the Shift-IR state, and latched on the falling edge of TCK in the Update-IR state. The latched instruction value can be changed in the Update-IR and Test-Logic-Reset TAP controller states only. Synchronous entry into the test-logic-reset state results in the IDCODE instruction being loaded on the falling edge of TCK. Asynchronous entry into the test-logic-reset state results in asynchronous loading of the IDCODE instruction. During the capture-IR TAP controller state, the instruction shift register is loaded with the value 0b10101, making this value the register's read value when the TAP controller is sequenced into the Shift-IR state.

	0	1	2	3	4
R	1	0	1	0	1
W	Instruction Code				
Reset	0	0	0	0	1

Figure 35-3. 5-Bit Instruction Register

35.3.2 Bypass Register

The bypass register is a single-bit shift register path selected for serial data transfer between TDI and TDO when the BYPASS, CLAMP, HIGHZ, or reserve instructions are active. After entry into the capture-DR state, the single-bit shift register is set to a logic 0. Therefore, the first bit shifted out after selecting the bypass register is always a logic 0.

35.3.3 Device Identification Register

The device identification register, shown in Figure 35-4, allows the part revision number, design center, part identification number, and manufacturer identity code to be determined through the TAP. The device identification register is selected for serial data transfer between TDI and TDO when the IDCODE instruction is active. Entry into the capture-DR state while the device identification register is selected loads the IDCODE into the shift register to be shifted out on TDO in the Shift-DR state. No action occurs in the update-DR state.

IR[4:0]: 0_0001 (IDCODE) Access: User read only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	PRN				DC						PIN										MIC											ID
W																																
Reset[1]	*	*	*	*	1	0	0	0	0	0	1	0	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0	1	1	1	0	1

Figure 35-4. Device Identification Register

1 PRN default value is 0x0 for the device's initial mask set and changes for each mask set revision.

Table 35-1. Device Identification Register Field Descriptions

Field	Description
PRN	Part Revision Number. Contains the revision number of the device. This field changes with each revision of the device or module.
DC	Design Center. Indicates the Freescale design center. For the PXN20 family this value is 0x20.
PIN	Part Identification Number. Contains the part number of the device. For the PXN20 family, this value is 0x268.
MIC	Manufacturer Identity Code. Contains the reduced Joint Electron Device Engineering Council (JEDEC) ID for Freescale, 0xE.
ID	IDCODE Register ID. Identifies this register as the device identification register and not the bypass register. Always set to 1.

35.3.4 Boundary Scan Register

The boundary scan register is connected between TDI and TDO when the EXTEST, SAMPLE, or SAMPLE/PRELOAD instructions are active. It is used to capture input pin data, force fixed values on output pins, and select a logic value and direction for bidirectional pins. Each bit of the boundary scan register represents a separate boundary scan register cell, as described in the IEEE 1149.1-2001 standard and discussed in Section 35.4.5, Boundary Scan.

35.4 Functional Description

35.4.1 JTAGC Reset Configuration

While in reset, the TAP controller is forced into the test-logic-reset state, thus disabling the test logic and allowing normal operation of the on-chip system logic. The instruction register is also loaded with the IDCODE instruction.

35.4.2 IEEE 1149.1-2001 (JTAG) Test Access Port

The JTAGC uses the IEEE 1149.1-2001 TAP for accessing registers. This port can be shared with other TAP controllers on the MCU. Ownership of the port is determined by the value of the JCOMP signal and the currently loaded instruction. For more detail on TAP sharing via JTAGC instructions refer to Section 35.4.4.2, ACCESS_AUX_TAP_x Instructions.

Data is shifted between TDI and TDO though the selected register starting with the least significant bit, as illustrated in Figure 35-5. This applies for the instruction register, test data registers, and the bypass register.

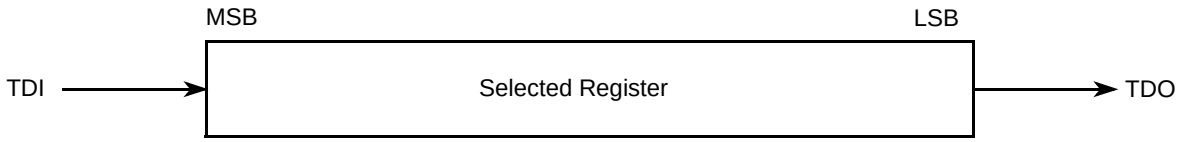
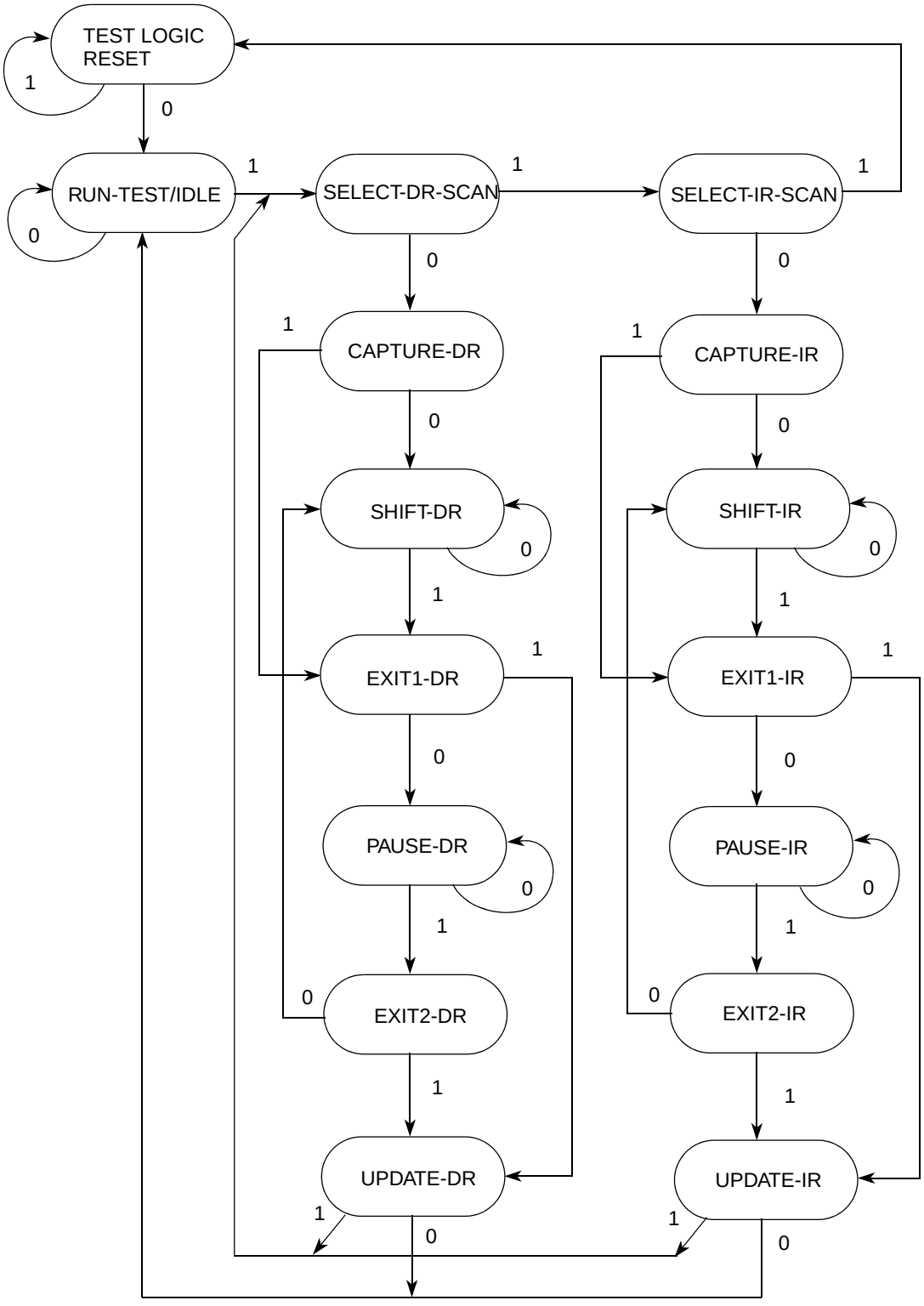


Figure 35-5. Shifting Data Through a Register

35.4.3 TAP Controller State Machine

The TAP controller is a synchronous state machine that interprets the sequence of logical values on the TMS pin. Figure 35-6 shows the machine's states. The value shown next to each state is the value of the TMS signal sampled on the rising edge of the TCK signal. As Figure 35-6 shows, holding TMS at logic 1 while clocking TCK through a sufficient number of rising edges also causes the state machine to enter the test-logic-reset state.



NOTE: The value shown adjacent to each state transition in this figure represents the value of TMS at the time of a rising edge of TCK.

Figure 35-6. IEEE 1149.1-2001 TAP Controller Finite State Machine

35.4.3.1 Enabling the TAP Controller

The JTAGC TAP controller is enabled by setting JCOMP to a logic 1 value.

35.4.3.2 Selecting an IEEE 1149.1-2001 Register

Access to the JTAGC data registers is achieved by loading the instruction register with any of the JTAGC instructions while the JTAGC is enabled. Instructions are shifted in via the select-IR-scan path and loaded in the update-IR state. At this point, all data register access is performed via the select-DR-scan path.

The select-DR-scan path is used to read or write the register data by shifting in the data (lsb first) during the shift-DR state. When reading a register, the register value is loaded into the IEEE 1149.1-2001 shifter during the capture-DR state. When writing a register, the value is loaded from the IEEE 1149.1-2001 shifter to the register during the update-DR state. When reading a register, there is no requirement to shift out the entire register contents. Shifting may be terminated after the required number of bits have been acquired.

35.4.4 JTAGC Instructions

The JTAGC implements the IEEE 1149.1-2001 defined instructions listed in Table 35-2. This section gives an overview of each instruction. Refer to the IEEE 1149.1-2001 standard for more details.

Table 35-2. JTAG Instructions

Instruction	Code[4:0]	Instruction Summary
IDCODE	00001	Selects device identification register for shift
SAMPLE/PRELOAD	00010	Selects boundary scan register for shifting, sampling, and preloading without disturbing functional operation
SAMPLE	00011	Selects boundary scan register for shifting and sampling without disturbing functional operation
EXTEST	00100	Selects boundary scan register while applying preloaded values to output pins and asserting functional reset
HIGHZ	01001	Selects bypass register while three-stating all output pins and asserting functional reset
CLAMP	01100	Selects bypass register while applying preloaded values to output pins and asserting functional reset
ACCESS_AUX_TAP_NPC	10000	Grants the Nexus port controller (NPC) ownership of the TAP
ACCESS_AUX_TAP_Z6 (from _ONCE)	10001	Grants the Nexus e200z6 core interface ownership of the TAP
ACCESS_AUX_TAP_Z0	11001	Grants the Nexus e200z0 core interface ownership of the TAP
ACCESS_AUX_TAP_MULTI	11100	Daisy chaining the e200z6 and e200z0 cores—allows instructions to be clocked into both the e200z0 and e200z6 serially
BYPASS	11111	Selects bypass register for data operations

Table 35-2. JTAG Instructions (continued)

Instruction	Code[4:0]	Instruction Summary
Factory Debug Reserved[1]	00010 00011 00100 00101 00110 00111 01001 01010 01100	Intended for factory debug only
Reserved[2]	All Other Codes	Decoded to select bypass register

1 Intended for factory debug, and not customer use

2 Freescale reserves the right to change the decoding of reserved instruction codes in the future

35.4.4.1 BYPASS Instruction

BYPASS selects the bypass register, creating a single-bit shift register path between TDI and TDO. BYPASS enhances test efficiency by reducing the overall shift path when no test operation of the MCU is required. This allows more rapid movement of test data to and from other components on a board that are required to perform test functions. While the BYPASS instruction is active, the system logic operates normally.

35.4.4.2 ACCESS_AUX_TAP_*x* Instructions

The ACCESS_AUX_TAP_*x* instructions allow the Nexus modules on the MCU to take control of the TAP. When this instruction is loaded, control of the TAP pins is transferred to the selected auxiliary TAP controller. Any data input via TDI and TMS is passed to the selected TAP controller, and any TDO output from the selected TAP controller is sent back to the JTAGC to be output on the pins. The JTAGC regains control of the JTAG port during the UPDATE-DR state if the PAUSE-DR state was entered. Auxiliary TAP controllers are held in RUN-TEST/IDLE while they are inactive.

See Section 35.5, e200z0 and e200z6 OnCE Controllers, for a block diagram and e200z0 OnCE controller register descriptions.

35.4.4.3 CLAMP Instruction

CLAMP allows the state of signals driven from MCU pins to be determined from the boundary scan register while the bypass register is selected as the serial path between TDI and TDO. CLAMP enhances test efficiency by reducing the overall shift path to a single bit (the bypass register) while conducting an EXTEST type of instruction through the boundary scan register. CLAMP also asserts the internal system reset for the MCU to force a predictable internal state.

35.4.4.4 EXTEST—External Test Instruction

EXTEST selects the boundary scan register as the shift path between TDI and TDO. It allows testing of off-chip circuitry and board-level interconnections by driving preloaded data contained in the boundary scan register onto the system output pins. Typically, the preloaded data is loaded into the boundary scan register using the SAMPLE/PRELOAD instruction before the selection of EXTEST. EXTEST asserts the internal system reset for the MCU to force a predictable internal state while performing external boundary scan operations.

35.4.4.5 HIGHZ Instruction

HIGHZ selects the bypass register as the shift path between TDI and TDO. While HIGHZ is active, all output drivers are placed in an inactive drive state (for example, high impedance). HIGHZ also asserts the internal system reset for the MCU to force a predictable internal state.

35.4.4.6 IDCODE Instruction

IDCODE selects the 32-bit device identification register as the shift path between TDI and TDO. This instruction allows interrogation of the MCU to determine its version number and other part identification data. IDCODE is the instruction placed into the instruction register when the JTAGC is reset.

35.4.4.7 SAMPLE Instruction

The SAMPLE instruction obtains a sample of the system data and control signals present at the MCU input pins and immediately before the boundary scan register cells at the output pins. This sampling occurs on the rising edge of TCK in the capture-DR state when the SAMPLE instruction is active. The sampled data is viewed by shifting it through the boundary scan register to the TDO output during the Shift-DR state. There is no defined action in the update-DR state. Both the data capture and the shift operation are transparent to system operation.

35.4.4.8 SAMPLE/PRELOAD Instruction

The SAMPLE/PRELOAD instruction has two functions:

- First, the SAMPLE portion of the instruction obtains a sample of the system data and control signals present at the MCU input pins and immediately before the boundary scan register cells at the output pins. This sampling occurs on the rising edge of TCK in the capture-DR state when the SAMPLE/PRELOAD instruction is active. The sampled data is viewed by shifting it through the boundary scan register to the TDO output during the shift-DR state. The data capture and the shift operation are transparent to system operation.
- Secondly, the PRELOAD portion of the instruction initializes the boundary scan register cells before selecting the EXTEST or CLAMP instructions to perform boundary scan tests. This is achieved by shifting in initialization data to the boundary scan register during the shift-DR state. The initialization data is transferred to the parallel outputs of the boundary scan register cells on the falling edge of TCK in the update-DR state. The data is applied to the external output pins by the EXTEST or CLAMP instruction. System operation is not affected.

35.4.5 Boundary Scan

The boundary scan technique allows signals at component boundaries to be controlled and observed through the shift-register stage associated with each pad. Each stage is part of a larger boundary scan register cell, and cells for each pad are interconnected serially to form a shift-register chain around the border of the design. The boundary scan register consists of this shift-register chain, and is connected between TDI and TDO when the EXTEST, SAMPLE, or SAMPLE/PRELOAD instructions are loaded. The shift-register chain contains a serial input and serial output, as well as clock and control signals.

35.5 e200z0 and e200z6 OnCE Controllers

The e200z0 core OnCE controller supports a complete set of Nexus 1 debug features, as well as providing access to the Nexus2+ configuration registers. A complete discussion of the e200z0 OnCE debug features is available in the *e200z0 Reference Manual*.

The following sections describe functionality of the e200z0 OnCE controller; however, the e200z6 OnCE controller operates in the same manner as the e200z0 OnCE controller, and is fully documented in the *e200z6 Reference Manual*.

NOTE

The register select field in the e200z6 OnCE command register (OCMD[RS]) does not implement the shared nexus control register (SNC).

35.5.1 e200z0 OnCE Controller Block Diagram

Figure 35-7 is a block diagram of the e200z0 OnCE block.

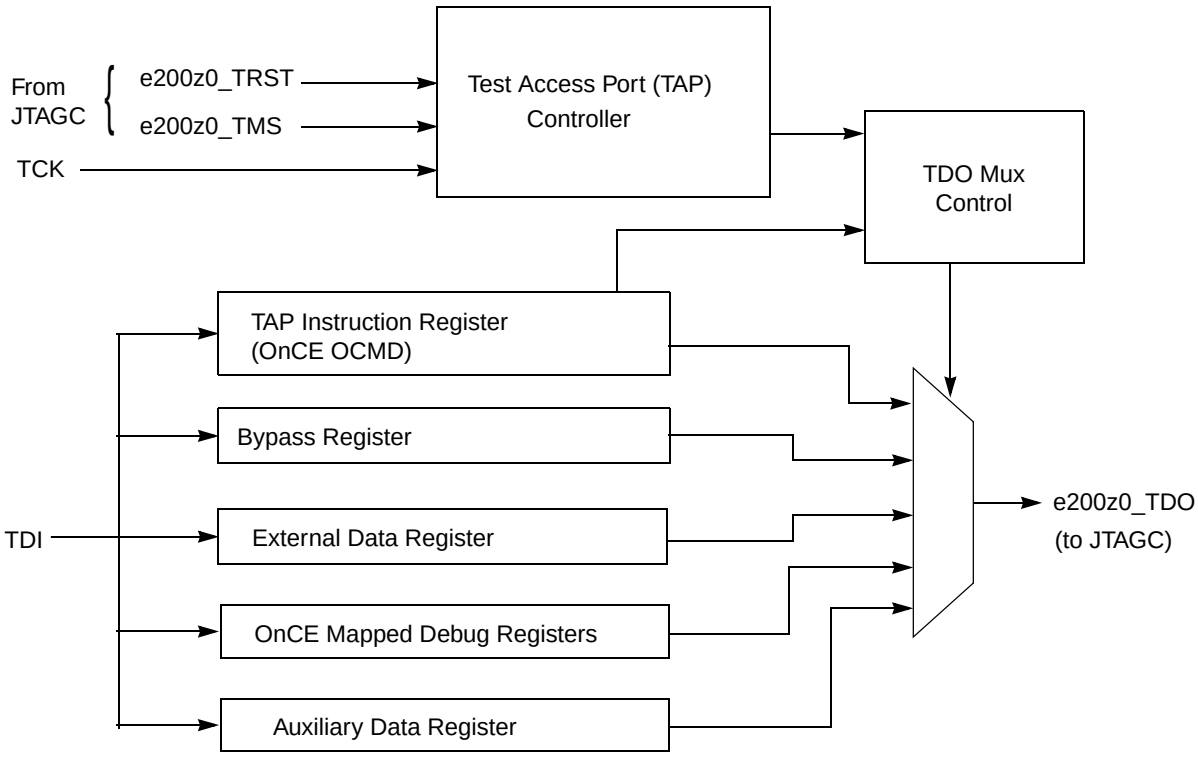


Figure 35-7. e200z0 OnCE Block Diagram

35.5.2 e200z0 OnCE Controller Functional Description

The functional description for the e200z0 OnCE controller is the same as for the JTAGC, with the differences described below.

35.5.2.1 Enabling the TAP Controller

To access the e200z0 OnCE controller, the proper JTAGC instruction needs to be loaded in the JTAGC instruction register, as discussed in Section 35.1.3.4, TAP Sharing Mode. The e200z0 OnCE TAP controller may either be accessed independently or chained with the e200z6 OnCE TAP controller, such that the TDO output of the e200z6 TAP controller is fed into the TDI input of the e200z0 TAP controller. The chained configuration allows commands to be loaded into both core's OnCE registers in one shift operation, so that both cores can be sent a GO command at the same time for example.

35.5.3 e200z0 OnCE Controller Register Descriptions

Most e200z0 OnCE debug registers are fully documented in the *e200z0 Reference Manual*.

35.5.3.1 OnCE Command Register (OCMD)

The OnCE command register (OCMD) is a 10-bit shift register that receives its serial data from the TDI pin and serves as the instruction register (IR). It holds the 10-bit commands to be used as input for the e200z0 OnCE Decoder. The OCMD is shown in Figure 35-8. The OCMD is updated when the TAP

controller enters the update-IR state. It contains fields for controlling access to a resource, as well as controlling single-step operation and exit from OnCE mode.

Although the OCMD is updated during the update-IR TAP controller state, the corresponding resource is accessed in the DR scan sequence of the TAP controller, and as such, the update-DR state must be transitioned through in order for an access to occur. In addition, the update-DR state must also be transitioned through in order for the single-step and/or exit functionality to be performed, even though the command appears to have no data resource requirement associated with it.

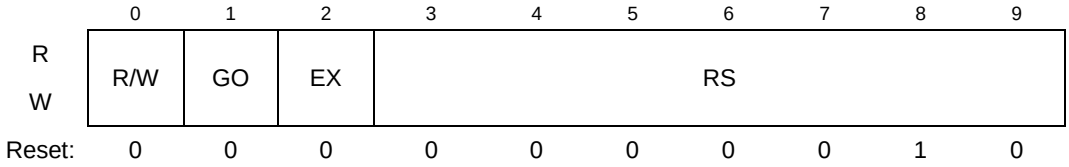


Figure 35-8. OnCE Command Register (OCMD)

Table 35-3. e200z0 and e200z6 OnCE Register Addressing

RS	Register Selected
000 0000 – 000 0001	Reserved
000 0010	JTAG ID (read-only)
000 0011 – 000 1111	Reserved
001 0000	CPU Scan Register (CPUSCR)
001 0001	No Register Selected (Bypass)
001 0010	OnCE Control Register (OCR)
001 0011 – 001 1111	Reserved
010 0000	Instruction Address Compare 1 (IAC1)
010 0001	Instruction Address Compare 2 (IAC2)
010 0010	Instruction Address Compare 3 (IAC3)
010 0011	Instruction Address Compare 4 (IAC4)
010 0100	Data Address Compare 1 (DAC1)
010 0101	Data Address Compare 2 (DAC2)
010 0110 – 010 1011	Reserved
010 1100	Debug Counter Register (DBCNT)[1]
010 1101	Debug PCFIFO (PCFIFO) (read-only)[1]
010 1110 – 010 1111	Reserved
011 0000	Debug Status Register (DBSR)
011 0001	Debug Control Register 0 (DBCR0)
011 0010	Debug Control Register 1 (DBCR1)
011 0011	Debug Control Register 2 (DBCR2)

Table 35-3. e200z0 and e200z6 OnCE Register Addressing (continued)

RS	Register Selected
011 0100	Debug Control Register 3 (DBCR3)[2]
011 0101 – 110 1111	Reserved (do not access)
111 0000 – 111 1001	General Purpose Register Selects [0:9]
111 1010	Cache Debug Access Control Register (CDACNTL)[1]
111 1011	Cache Debug Access Data Register (CDADATA)[1]
111 1100 – 111 1011	Reserved
111 1100	Nexus Access
111 1101	LSRL Select (factory test use only)
111 1110	Enable_OnCE
111 1111	Bypass

1 Reserved, not implemented on e200z0.

2 Reserved, not implemented on e200z0. Do not access.

35.6 Initialization/Application Information

The test logic is a static logic design, and TCK can be stopped in either a high or low state without loss of data. However, the system clock is not synchronized to TCK internally. Any mixed operation using both the test logic and the system functional logic requires external synchronization.

To initialize the JTAGC module and enable access to registers, the following sequence is required:

1. Set the JCOMP signal to logic 1, thereby enabling the JTAGC TAP controller.
2. Load the appropriate instruction for the test or action to be performed.

Chapter 36
Nexus Development Interface (NDI)

36.1 Introduction

NOTE

The Power PC standard is to number the register bits according to the MSB = 0 convention. However, the Nexus standard is to number the register bits according to the LSB = 0 convention.

Register bits in this chapter are numbered according to the Nexus standard (LSB = 0 convention).

The PXN20 device contains multiple Nexus clients that communicate over a single IEEE®-ISTO 5001™-2003 Nexus class 3+ and 2+ combined JTAG IEEE 1149.1/auxiliary out interface. Combined, all of the Nexus clients are referred to as the Nexus development interface (NDI). Class 3+ Nexus allows for program, data, and ownership trace of the microcontroller execution without access to the external data and address buses. Class 2+ Nexus allows for program and ownership trace of the microcontroller execution without access to the external data and address buses.

Communication to the NDI is handled via the auxiliary port and the JTAG port.

- The auxiliary port is comprised of 16 output pins and 1 input pin. The output pins include 1 message clock out (MCKO) pin, 12 message data out (MDO) pins, 2 message start/end out ($\overline{\text{MSEO}}$) pins, and 1 event out ($\overline{\text{EVTO}}$) pin. Event in ($\overline{\text{EVTI}}$) is the only input pin for the auxiliary port.
- The JTAG port consists of four inputs and one output. These pins include JTAG compliance select (JCOMP), test data input (TDI), test data output (TDO), test mode select (TMS), and test clock input (TCK). TDI, TDO, TMS, and TCK are compliant with the IEEE 1149.1-2001 standard and are shared with the NDI through the test access port (TAP) interface. JCOMP along with power-on reset and the TAP state machine are used to control reset for the NDI module. Ownership of the TAP is achieved by loading the appropriate enable instruction for the desired Nexus client in the JTAG controller (JTAGC) when JCOMP is asserted. See Section 36.4, Memory Map and Registers, for the JTAGC opcodes to access the different Nexus clients.

36.2 Block Diagram

Figure 36-1 shows a functional block diagram of the NDI. Figure 36-2 shows an implementation block diagram of the PXN20 NDI, which shows how the individual Nexus blocks are combined to form the NDI.

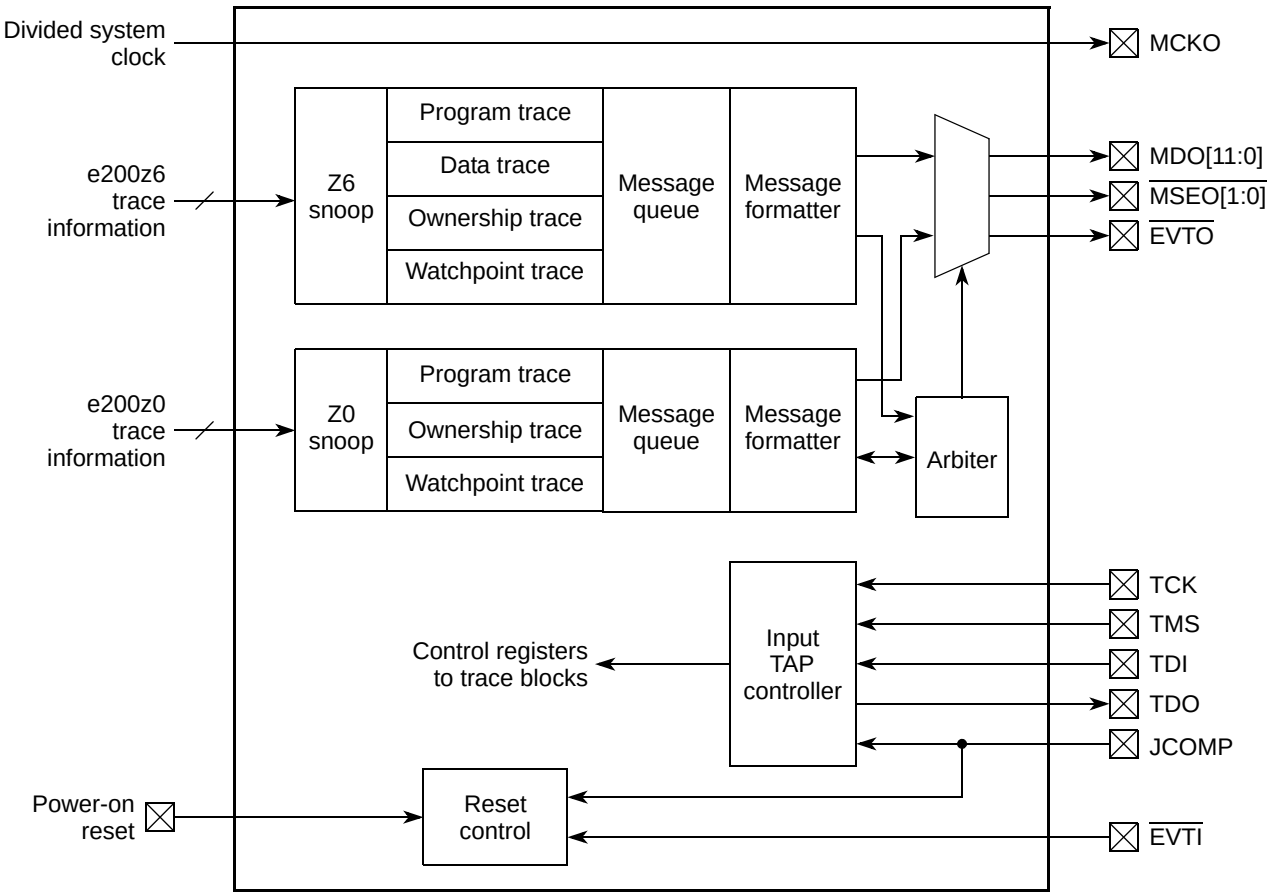


Figure 36-1. NDI Functional Block Diagram

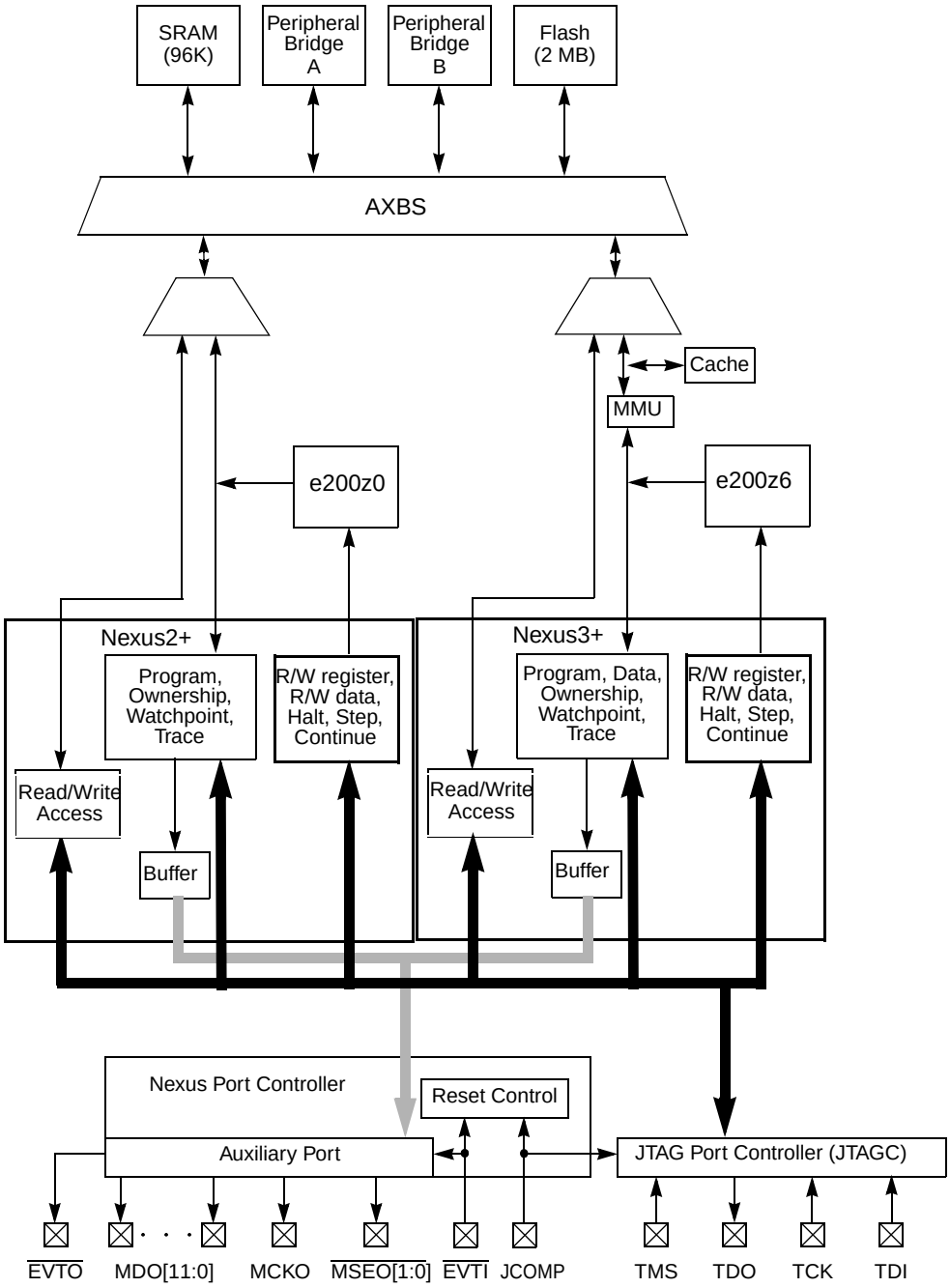


Figure 36-2. NDI Implementation Block Diagram

36.2.1 NDI Features

The NDI module of the PXN20 is compliant with the IEEE-ISTO 5001-2003 standard. The following features are implemented:

- 22-bit full duplex pin interface for medium and high visibility throughput
  - Only one mode is supported: full port mode (FPM). FPM comprises 12 MDO pins.
  - Auxiliary output port
    - One MCKO (message clock out) pin
    - 12 MDO (message data out) pins
    - Two $\overline{\text{MSEO}}$ (message start/end out) pins
    - One $\overline{\text{EVTO}}$ (event out) pin
  - Auxiliary input port uses one $\overline{\text{EVTI}}$ (event in) pin
  - Five-pin JTAG port (JCOMP, TDI, TDO, TMS, and TCK)
- The NPC block performs the following functions:
  - Controls arbitration for ownership of the Nexus Auxiliary Output Port
  - Nexus Device Identification Register and Messaging
  - Generates MCKO enable and frequency division control signals
  - Controls sharing of EVTO
  - Generates an MCKO clock gating control signal to enable gating of MCKO when the auxiliary output port is idle.
  - Control of the device-wide debug mode
  - Generates asynchronous reset signal for Nexus blocks based on JCOMP input and power-on reset status
- Host processor (e200z6) development support features (Nexus3+)
  - IEEE-ISTO 5001-2003 standard class 3 compliant.
  - Program trace via branch trace messaging (BTM). Branch trace messaging displays program flow discontinuities (direct branches, indirect branches, exceptions, etc.), allowing the development tool to interpolate what transpires between the discontinuities. Thus, static code may be traced.
  - Data trace via data write messaging (DWM) and data read messaging (DRM). This allows the development tool to trace reads and/or writes to selected internal memory resources.
  - Ownership trace via ownership trace messaging (OTM). OTM facilitates ownership trace by providing visibility of which process ID or operating system task is activated. An ownership trace message is transmitted when a new process/task is activated, allowing development tools to trace ownership flow.
  - Run-time access to the on-chip memory map via the JTAG port. This allows for enhanced download/upload capabilities.
  - Watchpoint messaging (WPM) via the auxiliary port.
  - Watchpoint trigger enable of program and/or data trace messaging.
  - Registers for Program Trace, Data Trace, Ownership Trace and Watchpoint Trigger.
  - All features controllable and configurable via JTAG port.
- I/O co-processor (e200z0) development support features (Nexus2+)

  - IEEE-ISTO 5001-2003 standard class 2 compliant with additional class 3 and 4 features available.
  - Program trace via branch trace messaging (BTM). Branch trace messaging displays program flow discontinuities (direct branches, indirect branches, exceptions, etc.), allowing the development tool to interpolate what transpires between the discontinuities. Thus, static code can be traced.
  - Ownership trace via ownership trace messaging (OTM). OTM facilitates ownership trace by providing visibility of which process ID or operating system task is activated. An ownership trace message is transmitted when a new process/task is activated, allowing development tools to trace ownership flow.
  - Run-time access to the embedded processor memory map via the JTAG port. This allows for enhanced download/upload capabilities.
  - Watchpoint messaging (WPM) via the auxiliary port.
  - Watchpoint trigger enable of program and/or data trace messaging.
  - Registers for Program Trace, Ownership Trace and Watchpoint Trigger.
  - All features controllable and configurable via JTAG port.
- The capability for an event out signal from either the e200z6 Nexus3+ or e200z0 Nexus2+ to generate a debug request to the other core, thus allowing both cores to enter debug mode within a short period of each other.

NOTE

Because the PXN20 implements multiple Nexus blocks, the configuration of the Message Data Out pins is controlled by the NPC.

36.2.2 Modes of Operation

The NDI block is in reset when the TAP controller state machine is in the TEST-LOGIC-RESET state. The TEST-LOGIC-RESET state is entered on the assertion of the power-on reset signal, negation of JCOMP, or through state machine transitions controlled by TMS. Assertion of JCOMP allows the NDI to move out of the reset state, and is a prerequisite to grant Nexus clients control of the TAP. Ownership of the TAP is achieved by loading the appropriate enable instruction for the desired Nexus client in the JTAG controller (JTAGC) block when JCOMP is asserted.

The NPC transitions out of the reset state immediately following negation of power-on reset.

36.2.2.1 Nexus Reset Mode

In Nexus reset mode, the following actions occur:

- Register values default back to their reset values.
- The message queues are marked as empty.
- The auxiliary output port pins are negated if the NDI controls the pads.
- The TDO output buffer is disabled if the NDI has control of the TAP.
- The TDI, TMS, and TCK inputs are ignored.

- The NDI block indicates to the MCU that it is not using the auxiliary output port. This indication can be used to three-state the output pins or use them for another function.

36.2.2.2 Full-Port Mode

In full-port mode, all available MDO pins are used to transmit messages. All trace features are enabled or can be enabled by writing the configuration registers via the JTAG port. Twelve MDO pins are available in full-port mode.

36.2.2.3 Reduced-Port Mode

Reduced-port mode is not supported on the PXN20.

36.2.2.4 Disabled-Port Mode

In disabled-port mode, message transmission is disabled. Any debug feature that generates messages cannot be used. The primary features available are class 1 features and read/write access.

36.2.2.5 Censored Mode

The NDI supports internal flash censorship mode by preventing the transmission of trace messages and Nexus access to memory-mapped resources when censorship is enabled.

36.2.2.6 Halt Mode

Halt mode logic is implemented in the Nexus port controller (NPC). When a request is made to enter halt mode, the NDI block completes monitoring of any pending bus transaction, transmits all messages already queued, and acknowledges the halt request. After the acknowledgment, the system clock input are shut off by the clock driver on the device. While the clocks are shut off, the development tool cannot access the NDI. See Section 8.3.2.24, Halt Acknowledge Register (SIU_HLTACKn), for a description of Halt Mode entry.

36.3 External Signal Description

The auxiliary and JTAG pin interfaces provide for the transmission of messages from Nexus modules to the external development tools and for access to Nexus client registers. The auxiliary/JTAG pin definitions are outlined in Table 36-1.

Table 36-1. Signal Properties

Name	Port	Function
$\overline{\text{EVTO}}$	Auxiliary	Event Out pin
$\overline{\text{EVTI}}$	Auxiliary	Event In pin
MCKO	Auxiliary	Message Clock Out pin (from NPC)
MDO[11:0]	Auxiliary	Message Data Out pins
$\overline{\text{MSEO}}$[1:0]	Auxiliary	Message Start/End Out pins

Table 36-1. Signal Properties (continued)

Name	Port	Function
JCOMP	JTAG	JTAG Compliancy and TAP Sharing Control
TCK	JTAG	Test Clock Input
TDI	JTAG	Test Data Input
TDO	JTAG	Test Data Output
TMS	JTAG	Test Mode Select Input

Refer to Chapter 3, Signal Description, and Table 3-1 for detailed signal descriptions.

36.4 Memory Map and Registers

The NDI block contains no memory mapped registers. Nexus registers are accessed by the development tool via the JTAG port using a client select and a register index. The client select is controlled by loading the correct access instruction into the JTAG controller, as described in Section 35.4.4, JTAGC Instructions. After the desired client TAP is selected, OnCE registers for that client are accessible by loading the appropriate value in the RS[0:6] field of the OnCE command register (OCMD), as described in Section 35.5.3.1, OnCE Command Register (OCMD). Nexus is enabled, and the associated Nexus registers become accessible, by loading the NEXUS_ENABLE instruction into the RS[0:6] field of the OCMD. When Nexus register access is enabled, the desired Nexus register is accessible using the index shown in Table 36-3.

Detailed sequences for Nexus3+ register access are described in Section 36.6.10.8, IEEE 1149.1 (JTAG) RD/WR Sequences. Detailed sequences for Nexus2+ register access are described in Section 36.7.9.7, IEEE 1149.1 (JTAG) RD/WR Sequences.

Table 36-2. Nexus Client JTAG Instructions

Instruction	Description	Opcode
NPC JTAG Instruction Opcodes		
NEXUS_ENABLE	Opcode for NPC Nexus Enable instruction (4-bits)	0x0
BYPASS	Opcode for the NPC BYPASS instruction (4-bits)	0xF
e200z6 OnCE JTAG Instruction Opcodes[1]		
NEXUS3_ACCESS	Opcode for e200z6 OnCE Nexus Enable instruction (10-bits)	0x7C
BYPASS	Opcode for the e200z6 OnCE BYPASS instruction (10-bits)	0x7F
e200z0 OnCE JTAG Instruction Opcodes[2]		
NEXUS2_ACCESS	Opcode for e200z0 OnCE Nexus Enable instruction (10-bits)	0x7C
BYPASS	Opcode for the e200z0 OnCE BYPASS instruction (10-bits)	0x7F

1 Refer to the *e200z6 Reference Manual* for a complete list of available OnCE instructions.

2 Refer to the *e200z0 Reference Manual* for a complete list of available OnCE instructions.

Table 36-3. NDI Registers

Index	Register	Read Address[1]	Write Address[1]	Section/Page
NPC Registers[2]				
0x00	Device ID (DID)	0x00	—	36.5.4.3/36-15
0x7F	Port Configuration Register (PCR)	0x7F	0x7F	36.5.4.4/36-16
e200z6 Control/Status Registers[3]				
0x02	e200z6 Development Control1 (PPC_DC1)	0x04	0x05	36.6.8.1/36-32
0x03	e200z6 Development Control2 (PPC_DC2)	0x06	0x07	36.6.8.1/36-32
0x04	e200z6 Development Status (PPC_DS)	0x08	—	36.6.8.2/36-34
0x07	e200z6 Read/Write Access Control/Status (Nexus3_RWCS)	0x0E	0x0F	36.6.8.3/36-35
0x09	e200z6 Read/Write Access Address (Nexus3_RWA)	0x12	0x13	36.6.8.4/36-36
0x0A	e200z6 Read/Write Access Data (Nexus3_RWD)	0x14	0x15	36.6.8.5/36-36
0x0B	e200z6 Watchpoint Trigger (PPC_WT)	0x16	0x17	36.6.8.6/36-38
0x0D	e200z6 Data Trace Control (PPC_DTC)	0x1A	0x1B	36.6.8.7/36-40
0x0E	e200z6 Data Trace Start Address 1 (PPC_DTSA1)	0x1C	0x1D	36.6.8.8/36-41
0x0F	e200z6 Data Trace Start Address 2 (PPC_DTSA2)	0x1E	0x1F	36.6.8.8/36-41
0x12	e200z6 Data Trace End Address 1 (PPC_DTEA1)	0x24	0x25	36.6.8.9/36-41
0x13	e200z6 Data Trace End Address 2 (PPC_DTEA2)	0x26	0x27	36.6.8.9/36-41
0x14 – 0x3F	Reserved	0x28–0x7E	0x29 – 0x7F	—
e200z0 Control/Status Registers[4]				
0x02	e200z0 Development Control1 (PPC_DC1)	0x04	0x05	36.7.7.1/36-76
0x03	e200z0 Development Control2 (PPC_DC2)	0x06	0x07	36.7.7.1/36-76
0x04	e200z0 Development Status (PPC_DS)	0x08	—	36.7.7.2/36-78
0x07	e200z0 Read/Write Access Control/Status (Nexus2_RWCS)	0x0E	0x0F	36.7.7.3/36-78
0x09	e200z0 Read/Write Access Address (Nexus2_RWA)	0x12	0x13	36.7.7.4/36-80
0x0A	e200z0 Read/Write Access Data (Nexus2_RWD)	0x14	0x15	36.7.7.5/36-80
0x0B	e200z0 Watchpoint Trigger (PPC_WT)	0x16	0x17	36.7.7.6/36-81
0x0C – 0x3F	Reserved	0x1A – 0x7E	0x19 – 0x7F	—

1 See Section 36.5.5.2.3, NPC IEEE 1149.1-2001 (JTAG) TAP, for a description of the read and write address usage for the e200z6 and e200z0 Nexus Control/Status registers.

2 See Section 36.5.4 for a detailed description of the NPC registers.

[3] See Section 36.6.8 for a detailed description of the e200z6 Nexus registers.
[4] See Section 36.7.7 for a detailed description of the e200z0 Nexus registers.

36.4.1 NDI Functional Description

The NDI block is implemented by integrating the following blocks on the PXN20:

- Nexus Port Controller Block
- Nexus e200z6 Development Interface (OnCE and Nexus3+ subblocks)
- Nexus e200z0 Development Interface (OnCE and Nexus2+ subblocks)

Refer to the block guides for more information about these blocks. Note that the TAP controller logic, reset logic, and some miscellaneous logic are duplicated in all these blocks.

36.4.1.1 Enabling Nexus Clients for TAP Access

Once the NDI is out of the reset state, the loading of a specific instruction in the JTAG controller (JTAGC) block is required to grant the NDI ownership of the TAP. Each Nexus client has its own JTAGC instruction opcode for ownership of the TAP, granting that client the means to read/write its registers. The JTAGC instruction opcode for each Nexus client is shown in Table 35-2. Once the JTAGC opcode for a client has been loaded, the client is enabled by loading its NEXUS-ENABLE instruction. Opcodes for all other instructions supported by Nexus clients can be found in the relevant sections of this chapter.

Details for accessing the NPC registers is covered in Section 36.5.5.2.3, NPC IEEE 1149.1-2001 (JTAG) TAP.

36.4.1.2 TAP Sharing

Each of the individual Nexus blocks on the MCU implements a TAP controller for accessing its registers. The JTAGC controls the ownership of the TAP so that the interface to all of these individual TAP controllers appears to be a single port from outside the device. Once a Nexus client has been granted ownership of the TAP, any data input via TDI and TMS is passed to the selected TAP controller, and any TDO output from the selected TAP controller is sent back to the JTAGC to be output on the pins. The JTAGC regains control of the JTAG port during the UPDATE-DR state if the PAUSE-DR state was entered. Auxiliary TAP controllers are held in RUN-TEST/IDLE while they are inactive.

36.4.1.3 Configuring the NDI for Nexus Messaging

The NDI is placed in disabled mode upon exit of power-on reset. If message transmission via the auxiliary port is desired, a write to the port configuration register (PCR) located in the NPC is then required to enable the NDI and select the mode of operation. Asserting MCKO_EN in the PCR places the NDI in enabled mode and enables MCKO. The frequency of MCKO is selected by writing the MCKO_DIV field. Asserting the FPM bit selects full-port mode.

NOTE

Reduced-port mode is not supported. The FPM bit in the PCR must be set when configuring the NDI.

When writing to the PCR, the PCR LSB must be written to a logic 0. Setting the LSB of the PCR enables factory debug mode and prevents the transmission of Nexus messages.

Table 36-4 describes the NDI configuration options.

Table 36-4. NDI Configuration Options

JCOMP Asserted	MCKO_EN bit of PCR	FPM bit of PCR[1]	Configuration
No	X	X	Reset
Yes	0	X	Disabled
Yes	1	1	Full-Port Mode

1 FPM must always be set.

36.4.1.4 Programmable MCKO Frequency

MCKO is an output clock to the development tools used for the timing of $\overline{\text{MSEO}}$ and MDO pin functions. MCKO is derived from the system clock, and its frequency is determined by the value of the MCKO_DIV field in the port configuration register (PCR) located in the NPC. Possible operating frequencies include SYS_CLK, one-half, one-quarter, and one-eighth SYS_CLK speed.

Table 36-5 shows the MCKO_DIV encodings. In this table, SYS_CLK represents the system clock frequency.

Table 36-5. MCKO_DIV Values

MCKO_DIV[2:0]	MCKO Frequency
0b000[1]	SYS_CLK
0b001	SYS_CLK/2
0b010	Reserved
0b011	SYS_CLK/4
0b100	Reserved
0b101	Reserved
0b110	Reserved
0b111	SYS_CLK/8

1 The SYS_CLK setting for MCKO frequency should only be used if this setting does not violate the maximum operating frequency of the auxiliary port pins.

36.4.1.5 Nexus Messaging

Most of the messages transmitted by the NDI include a SRC field. This field is used to identify which source generated the message. Table 36-6 shows the values used for the SRC field by the different clients on the device. These 4-bit values are specific to the device.

Table 36-6. SRC Packet Encodings

SRC[3:0]	Client
0b0000	e200z6
0b0001- 0b0111	Reserved
0b1000	e200z0
0b1001- 0b1111	Reserved

36.4.1.6 e200z6 and e200z0 Cross Triggering Control

To enable a debug event in one core to cause a debug event in the other core at approximately the same time, the $\overline{\text{EVTO}}$ signal from the e200z0 Nexus2+ or e200z6 Nexus3+ is connected to the other core's devt2 input. When enabled in each core's Nexus1 DBCR0 register, a pulse of the devt2 signal causes a debug event to occur. In this case, only one external $\overline{\text{EVTO}}$ signal is generated and each core controls whether or not $\overline{\text{EVTO}}$ causes a debug event to occur.

Interconnection of debug mode control signals are shown in Figure 36-3.

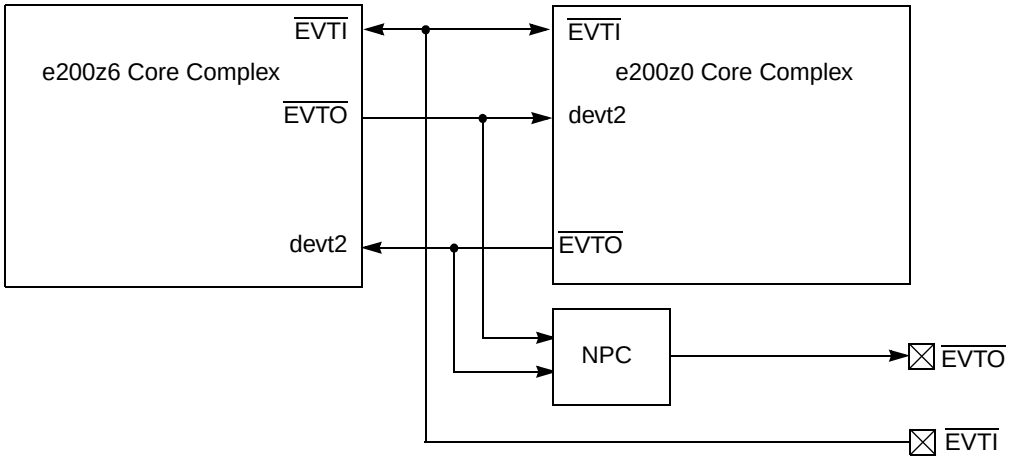


Figure 36-3. Debug Mode Control Interconnections

Figure 36-4 shows the flow for configuring the e200z0 Nexus2+ to cause an debug request to the e200z6.

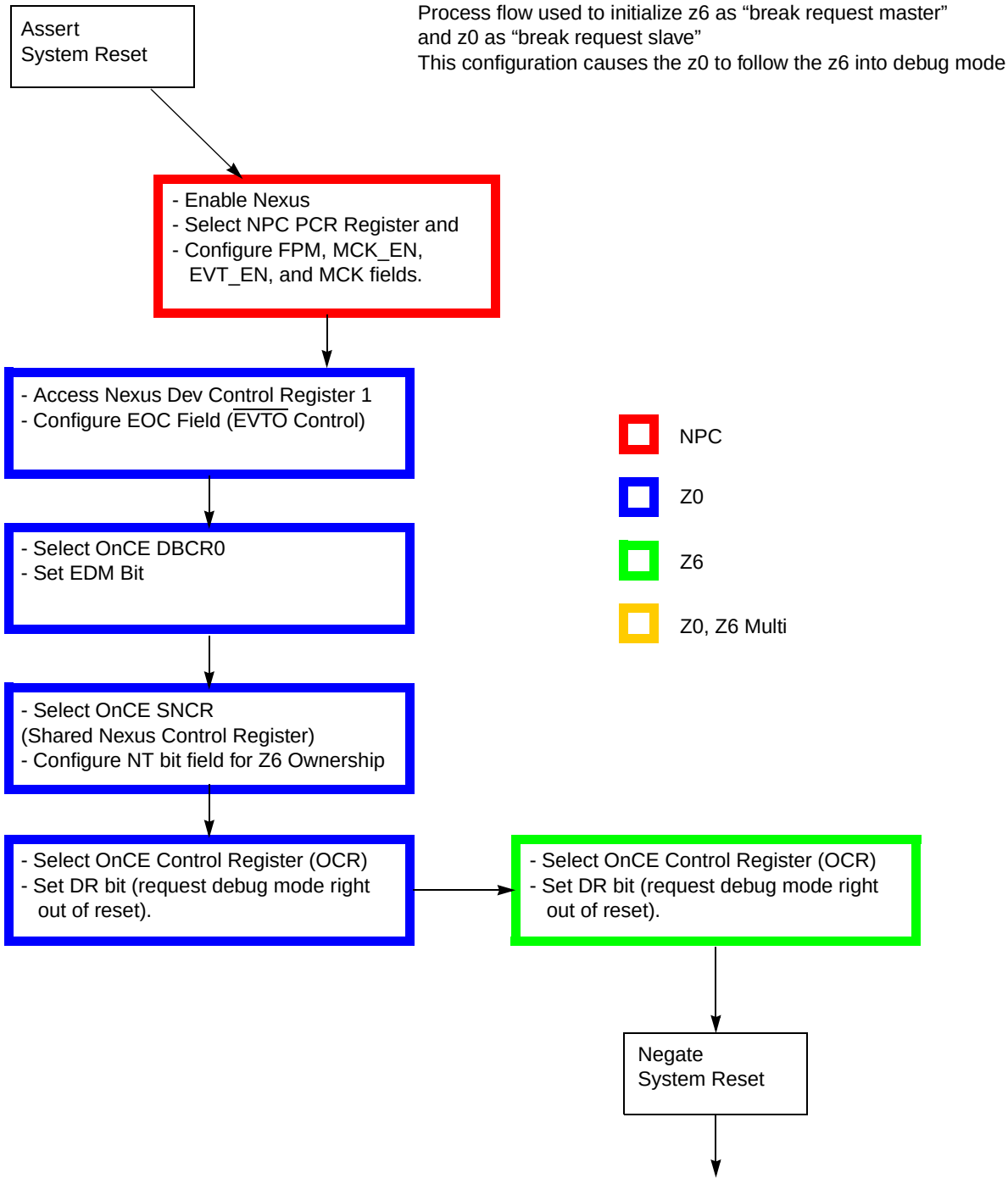


Figure 36-4. Flow for Enabling e200z0 to put the e200z6 into Debug Mode

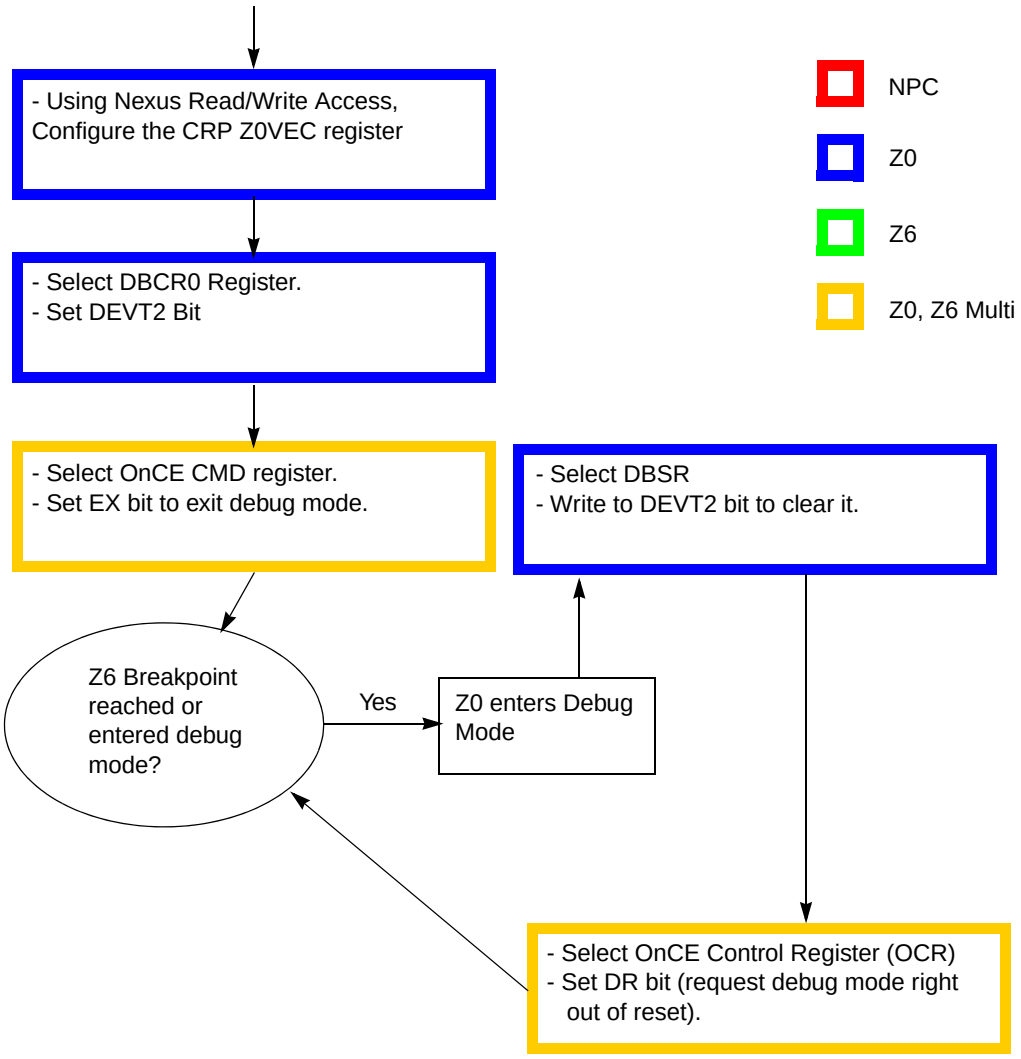


Figure 36-4. Flow for Enabling e200z0 to put the e200z6 into Debug Mode (Continued)

The flow for enabling the e200z6 Nexus3+ to cause a debug request to the e200z0 is similar.

36.5 Nexus Port Controller (NPC)

The Nexus Port Controller (NPC) is that part of the NDI that controls access and arbitration of the device's internal Nexus modules. The NPC contains the port configuration register (PCR) and the device identification register (DID). The contents of the NPC DID are the same as the JTAGC device identification register.

36.5.1 NPC Overview

The PXN20 incorporates multiple modules that require development support. Each of these modules implements a development interface based on the IEEE-ISTO 5001-2001 standard and must share the input and output ports that interface with the development tool. The NPC controls the usage of these ports

in a manner that allows the individual modules to share the ports, while appearing to the development tool as a single module.

36.5.2 NPC Features

The NPC performs the following functions:

- Controls arbitration for ownership of the Nexus auxiliary output port
- Nexus device identification register and messaging
- Generates MCKO enable and frequency division control signals
- Controls sharing of $\overline{\text{EVTO}}$

36.5.3 Control of the device-wide debug mode NPC Memory Map

Table 36-7 shows the NPC registers by index values. The registers are not memory-mapped and can only be accessed via the TAP. The NPC does not implement the client select control register because the value does not matter when accessing the registers. Note that the bypass register (refer to Section 36.5.4.1, Bypass Register) and instruction register (refer to Section 36.5.4.2, Instruction Register) have no index values. These registers are not accessed in the same manner as Nexus client registers.

Table 36-7. NPC Memory Map

Index	Register Name	Register Description	Size (bits)
0	DID	Device ID register	32
127	PCR	Port configuration register	32

36.5.4 NPC Register Descriptions

This section consists of NPC register descriptions.

36.5.4.1 Bypass Register

The bypass register is a single-bit shift register path selected for serial data transfer between TDI and TDO when the BYPASS instruction or any unimplemented instructions are active. After entry into the CAPTURE-DR state, the single-bit shift register is set to a logic 0. Therefore, the first bit shifted out after selecting the bypass register is always a logic 0.

36.5.4.2 Instruction Register

The NPC uses a 4-bit instruction register as shown in Figure 36-5. The instruction register is accessed via the SELECT_IR_SCAN path of the tap controller state machine, and allows instructions to be loaded into the module to enable the NPC for register access (NEXUS_ENABLE) or select the bypass register as the shift path from TDI to TDO (BYPASS or unimplemented instructions).

Instructions are shifted in through TDI while the TAP controller is in the Shift-IR state, and latched on the falling edge of TCK in the Update-IR state. The latched instruction value can only be changed in the Update-IR and test-logic-reset TAP controller states. Synchronous entry into the test-logic-reset state

results in synchronous loading of the BYPASS instruction. Asynchronous entry into the test-logic-reset state results in asynchronous loading of the BYPASS instruction. During the Capture-IR TAP controller state, the instruction register is loaded with the value of the previously executed instruction, making this value the register's read value when the TAP controller is sequenced into the Shift-IR state.

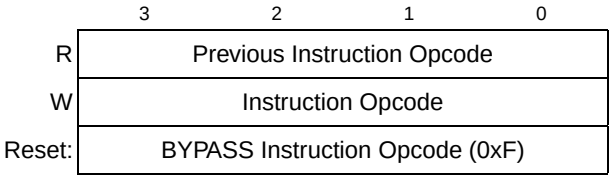


Figure 36-5. 4-Bit Instruction Register

36.5.4.3 Nexus Device ID Register (DID)

The NPC device identification register, shown in Figure 36-6, allows the part revision number, design center, part identification number, and manufacturer identity code of the device to be determined through the auxiliary output port, and serially through TDO. See Section 36.5.5.2.3, NPC IEEE 1149.1-2001 (JTAG) TAP. This register is read-only.

Reg Index: 0x00 Access: User read only

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	PRN				DC						PIN					
W																
Reset [1]	*	*	*	*	1	0	0	0	0	0	1	0	0	1	1	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	PIN (*continued*)				MIC											1
W																
Reset	1	0	0	0	0	0	0	0	0	0	0	1	1	1	0	1

Figure 36-6. Nexus Device ID Register (DID)

[1] Part Revision Number default value is 0x0 for the device's initial mask set and changes for each mask set revision.

Table 36-8. DID Field Descriptions

Field	Description
PRN	Part Revision Number. Contains the revision number of the part. This field changes with each revision of the device or module.
DC	Design Center. Indicates the Freescale design center. This value is 0x20.
PIN	Part Identification Number. Contains the part number of the device.
MIC	Manufacturer Identity Code. Contains the reduced Joint Electron Device Engineering Council (JEDEC) ID for Freescale, 0x00E.
bit 0	Fixed Per JTAG 1149.1. Always set to 1.

36.5.4.4 Port Configuration Register (PCR)

The PCR is used to select the NPC mode of operation, enable MCKO and select the MCKO frequency, and enable or disable MCKO gating. This register should be configured as soon as the NDI is enabled.

The PCR register may be rewritten by the debug tool subsequent to the enabling of the NPC for low power debug support. In this case, the debug tool may set and clear the LP_DBG_EN and SLEEP_SYNC bits, but must preserve the original state of the remaining bits in the register.

NOTE

The mode (MCKO_GT) or clock division (MCKO_DIV) bits must not be modified after MCKO has been enabled. Changing the mode or clock division while MCKO is enabled can produce unpredictable results.

Reg Index: 0x7F Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	FPM	MCKO_GT	MCKO_EN	MCKO_DIV			EVT_EN	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	LP_DBG_EN	0	0	0	0	0	SLEEP_SYNC	0	0	0	0	0	0	0	0	PSTAT_EN
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 36-7. Port Configuration Register (PCR)

Table 36-9. PCR Field Descriptions

Field	Description
FPM	Full Port Mode.The value of the FPM bit determines if the auxiliary output port uses the full MDO port or a reduced MDO port to transmit messages. 0 A subset of MDO pins are used to transmit messages. **Note:** All MDO pins are used to transmit messages.This bit must always be set when configuring the NDI.
MCKO_GT	MCKO Clock Gating Control.This bit is used to enable or disable MCKO clock gating. If clock gating is enabled, the MCKO clock is gated when the NPC is in enabled mode but not actively transmitting messages on the auxiliary output port. When clock gating is disabled, MCKO is allowed to run even if no auxiliary output port messages are being transmitted. 0 MCKO gating is disabled. 1 MCKO gating is enabled.
MCKO_EN	MCKO Enable. This bit enables the MCKO clock to run. When enabled, the frequency of MCKO is determined by the MCKO_DIV field. 0 MCKO clock is driven to zero. 1 MCKO clock is enabled.

Table 36-9. PCR Field Descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>MCKO_DIV</td><td>MCKO Division Factor. The value of this signal determines the frequency of MCKO relative to the system clock frequency when MCKO_EN is asserted. In this table, SYS_CLK represents the system clock frequency.
<table>
<tr><th>MCKO_DIV</th><th>MCKO Frequency</th></tr>
<tr><td>0b000[1]</td><td>SYS_CLK</td></tr>
<tr><td>0b001</td><td>SYS_CLK ÷ 2</td></tr>
<tr><td>0b010</td><td>Reserved</td></tr>
<tr><td>0b011</td><td>SYS_CLK ÷ 4</td></tr>
<tr><td>0b100</td><td>Reserved</td></tr>
<tr><td>0b101</td><td>Reserved</td></tr>
<tr><td>0b110</td><td>Reserved</td></tr>
<tr><td>0b111</td><td>SYS_CLK ÷ 8</td></tr>
</table>
1 The SYS_CLK setting for MCKO frequency should only be used if this setting does not violate the maximum operating frequency of the auxiliary port pins.</td></tr>
<tr><td>EVT_EN</td><td>$\overline{\text{EVTO}}$/$\overline{\text{EVTI}}$ Enable. This bit enables the $\overline{\text{EVTO}}$/$\overline{\text{EVTI}}$ port functions.
0 $\overline{\text{EVTO}}$/$\overline{\text{EVTI}}$ port disabled.
1 $\overline{\text{EVTO}}$/$\overline{\text{EVTI}}$ port enabled.</td></tr>
<tr><td>LP_DBG_EN</td><td>Low Power Debug Enable. The LP_DBG_EN bit enables debug functionality to support entry and exit from low power sleep mode.
0 Low power debug disabled.
1 Low power debug enabled.</td></tr>
<tr><td>SLEEP_SYNC</td><td>Sleep Mode Synchronization. The SLEEP_SYNC bit is used to synchronize the entry into / exit from sleep mode between the device and debug tool. Before entry into sleep mode, the device sets this bit. After reading SLEEP_SYNC as set, the debug tool then clears SLEEP_SYNC to acknowledge to the device that it may enter into sleep mode. During wakeup from sleep mode, the debug tool sets SLEEP_SYNC to acknowledge to the device that it may exit sleep mode.
0 No sleep mode entry pending.
1 Sleep mode entry pending.
Note: During sleep entry, the device sets the SLEEP_SYNC bit. The debug tool then clears this bit to enter low power mode. The bit changes to 0, but is not readable, since the device immediately enters low power mode. During sleep exit, the device then clears the SLEEP_SYNC bit. Then the debug tool sets this bit to exit low power mode. The bit does not change to 1.</td></tr>
<tr><td>PSTAT_EN</td><td>Processor status mode enable. Enables processor status (PSTAT) mode. In PSTAT mode, all auxiliary output port MDO pins are used to transmit processor status information, and Nexus messaging is unavailable.
0 PSTAT mode disabled.
1 PSTAT mode enabled.
Note: PSTAT mode is intended for factory processor debug only. Customers should clear the PSTAT_EN bit should be cleared to disable PSTAT mode. When PSTAT mode is enabled, no Nexus messages are transmitted under any circumstances.</td></tr>
</table>

36.5.5 NPC Functional Description

36.5.5.1 NPC Reset Configuration

The NPC is placed in disabled mode upon exit of reset. If message transmission via the auxiliary port is desired, a write to the PCR is then required to enable the NPC and select the mode of operation. Asserting MCKO_EN places the NPC in enabled mode and enables MCKO. The frequency of MCKO is selected by writing the MCKO_DIV field.

Table 36-4 describes the NPC reset configuration options.

36.5.5.2 Auxiliary Output Port

The auxiliary output port is shared by each of the Nexus modules on the device. The NPC communicates with each of the individual modules and arbitrates for access to the port. Additional information about the auxiliary port is found in Section 35.1.1.1, Individual and Multi-Core Debug.

36.5.5.2.1 Output Message Protocol

The protocol for transmitting messages via the auxiliary port is accomplished with the $\overline{\text{MSEO}}$ functions. The $\overline{\text{MSEO}}$ pins are used to signal the end of variable-length packets and the end of messages. They are not required to indicate the end of fixed-length packets. MDO and $\overline{\text{MSEO}}$ are sampled on the rising edge of MCKO.

Figure 36-8 illustrates the state diagram for $\overline{\text{MSEO}}$ transfers. All transitions not included in the figure are reserved, and must not be used.

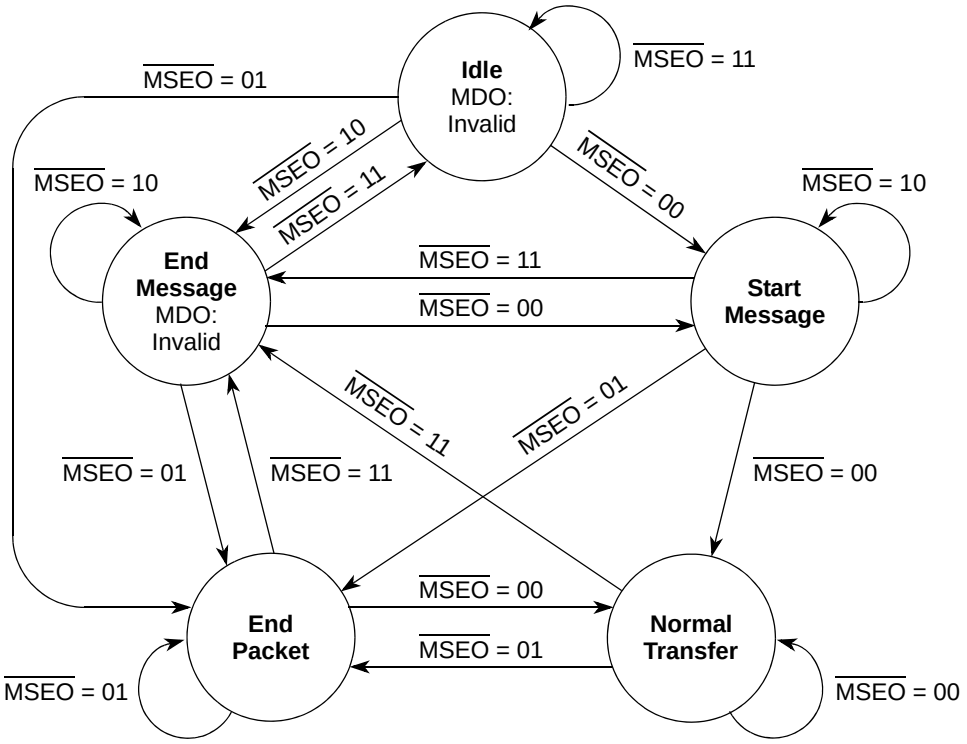


Figure 36-8. $\overline{\text{MSEO}}$ Transfers

36.5.5.2.2 Output Messages

In addition to sending out messages generated in other Nexus modules, the NPC can also output the device ID message contained in the device ID register on the MDO pins. The device ID message can also be sent out serially through TDO.

Table 36-10 describes the device ID message that the NPC can transmit on the auxiliary port. The TCODE is the first packet transmitted.

Table 36-10. NPC Output Messages

Message Name	Min. Packet Size (bits)	Max Packet Size (bits)	Packet Type	Packet Name	Packet Description
Device ID Message	6	6	Fixed	TCODE	Value = 1
	32	32	Fixed	ID	DID register contents

Figure 36-9 shows the various message formats that the pin interface formatter has to encounter.

Figure 36-9. Message Field Sizes

Message	TCODE	Field #1	Field #2	Field #3	Field #4	Field #5	Min. Size[1] (bits)	Max Size[2] (bits)
Device ID Message	1	Fixed = 32	NA	NA	NA	NA	38	38

1 Minimum information size. The actual number of bits transmitted depends on the number of MDO pins

2 Maximum information size. The actual number of bits transmitted depends on the number of MDO pins

The double edges in Figure 36-9 indicate the starts and ends of messages. Fields without shaded areas between them are grouped into super-fields and can be transmitted together without end-of-packet indications between them.

Rules of Messages

The rules of messages include the following:

- A variable-sized field within a message must end on a port boundary. (Port boundaries depend on the number of MDO pins active with the current reset configuration.)
- A variable-sized field may start within a port boundary only when following a fixed-length field.
- Super-fields must end on a port boundary.
- When a variable-length field is sized such that it does not end on a port boundary, it is necessary to extend and zero fill the remaining bits after the highest order bit so that it can end on a port boundary.
- Multiple fixed-length packets may start and/or end on a single clock.
- When any packet follows a variable-length packet, it must start on a port boundary.
- The field containing the TCODE number is always transferred out first, followed by subsequent fields of information.
- Within a field, the lowest significant bits are shifted out first. Figure 36-10 shows the transmission sequence of a message that is made up of a TCODE followed by three fields.

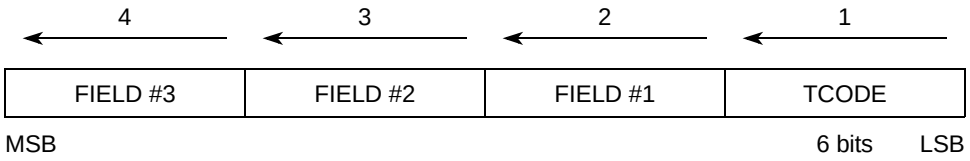


Figure 36-10. Transmission Sequence of Messages

Detailed message descriptions for the Nexus3+ module are described in Section 36.6.10. Detailed message descriptions for the Nexus2+ module are described in Section 36.7.9.

36.5.5.2.3 NPC IEEE 1149.1-2001 (JTAG) TAP

The NPC uses the IEEE 1149.1-2001 TAP, which uses the state machine shown in Figure 35-6 for accessing registers. The NPC also implements the Nexus controller state machine as defined by the IEEE-ISTO 5001-2003 standard as shown in Figure 36-11.

The instructions implemented by the NPC TAP controller are listed in Table 36-2.

Enabling the NPC TAP Controller

Assertion of the power-on reset signal, or negating JCOMP resets the NPC TAP controller. When not in power-on reset, the NPC TAP controller is enabled by asserting JCOMP and loading the ACCESS_AUX_TAP_NPC instruction in the JTAGC. Loading the NEXUS-ENABLE instruction then grants access to NPC registers.

Retrieving Device IDCODE

The Nexus TAP controller does not implement the IDCODE instruction. However, the device identification message can be output by the NPC through the auxiliary output port or shifted out serially by accessing the NPC device ID register through the TAP. If the NPC is enabled, transmission of the device identification message on the auxiliary output port MDO pins occurs immediately after a write to the PCR. Transmission of the device identification message serially through TDO is achieved by performing a read of the register contents.

Loading NEXUS-ENABLE Instruction

Access to the NPC registers is enabled by loading the NPC NEXUS-ENABLE instruction when NPC has ownership of the TAP. This instruction is shifted in via the SELECT-IR-SCAN path and loaded in the UPDATE-IR state. At this point, the Nexus controller state machine, shown in Figure 36-11, transitions to the REG_SELECT state. The Nexus controller has three states: idle, register select, and data access. Table 36-11 illustrates the IEEE® 1149.1 sequence to load the NEXUS-ENABLE instruction.

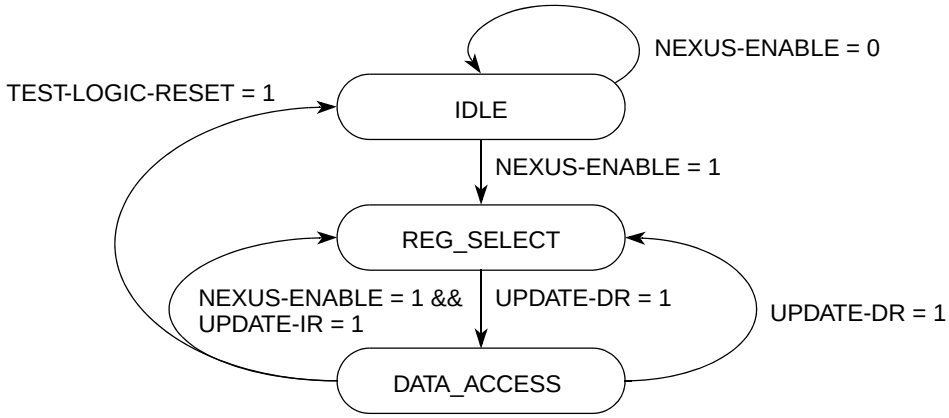


Figure 36-11. NEXUS Controller State Machine

Table 36-11. Loading NEXUS-ENABLE Instruction

Clock	TDI	TMS	IEEE 1149.1 State	Nexus State	Description
0	—	0	RUN-TEST/IDLE	IDLE	IEEE 1149.1-2001 TAP controller in idle state
1	—	1	SELECT-DR-SCAN	IDLE	Transitional state
2	—	1	SELECT-IR-SCAN	IDLE	Transitional state
3	—	0	CAPTURE-IR	IDLE	Internal shifter loaded with current instruction
4	—	0	SHIFT-IR	IDLE	TDO becomes active, and the IEEE® 1149.1-2001 shifter is ready. Shift in all but the last bit of the NEXUS_ENABLE instruction.
5-7	0	0	3 TCKS in SHIFT-IR	IDLE	
8	0	1	EXIT1-IR	IDLE	Last bit of instruction shifted in
9	—	1	UPDATE-IR	IDLE	NEXUS-ENABLE loaded into instruction register
10	—	0	RUN-TEST/IDLE	REG_SELECT	Ready to be read/write Nexus registers

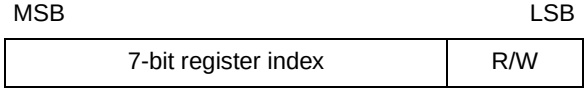
Selecting a Nexus Client Register

When the NEXUS-ENABLE instruction is decoded by the TAP controller, the input port allows development tool access to all Nexus registers. Each register has a 7-bit address index.

All register access is performed via the SELECT-DR-SCAN path of the IEEE® 1149.1–2001 TAP controller state machine. The Nexus controller defaults to the REG_SELECT state when enabled. Accessing a register requires two passes through the SELECT-DR-SCAN path: one pass to select the register and the second pass to read/write the register.

The first pass through the SELECT-DR-SCAN path is used to enter an 8-bit Nexus command consisting of a read/write control bit in the LSB followed by a 7-bit register address index, as illustrated in Figure 36-12. The read/write control bit is set to 1 for writes and 0 for reads.

Figure 36-12. IEEE® 1149.1 Controller Command Input



The second pass through the SELECT-DR-SCAN path is used to read or write the register data by shifting in the data (LSB first) during the SHIFT-DR state. When reading a register, the register value is loaded into the IEEE® 1149.1-2001 shifter during the CAPTURE-DR state. When writing a register, the value is loaded from the IEEE® 1149.1-2001 shifter to the register during the UPDATE-DR state. When reading a register, no requirement exists to shift out the entire register contents. Shifting may be terminated once the required number of bits have been acquired.

Table 36-12 illustrates a sequence that writes a 32-bit value to a register.

Table 36-12. Write to a 32-Bit Nexus Client Register

Clock	TMS	IEEE 1149.1 State	Nexus State	Description
0	0	RUN-TEST/IDLE	REG_SELECT	IEEE 1149.1-2001 TAP controller in idle state
1	1	SELECT-DR-SCAN	REG_SELECT	First pass through SELECT-DR-SCAN path
2	0	CAPTURE-DR	REG_SELECT	Internal shifter loaded with current value of controller command input.
3	0	SHIFT-DR	REG_SELECT	TDO becomes active, and write bit and 6 bits of register index shifted in.
7 TCKs				
11	1	EXIT1-DR	REG_SELECT	Last bit of register index shifted into TDI
12	1	UPDATE-DR	REG_SELECT	Controller decodes and selects register
13	1	SELECT-DR-SCAN	DATA_ACCESS	Second pass through SELECT-DR-SCAN path
14	0	CAPTURE-DR	DATA_ACCESS	Internal shifter loaded with current value of register
15	0	SHIFT-DR	DATA_ACCESS	TDO becomes active, and outputs current value of register while new value is shifted in through TDI
31 TCKs				
47	1	EXIT1-DR	DATA_ACCESS	Last bit of current value shifted out TDO. Last bit of new value shifted in TDI.
48	1	UPDATE-DR	DATA_ACCESS	Value written to register
49	0	RUN-TEST/IDLE	REG_SELECT	Controller returned to idle state. It could also return to SELECT-DR-SCAN to write another register.

36.5.5.2.4 Nexus Auxiliary Port Sharing

Each of the Nexus modules on the MCU implements a request/grant scheme to arbitrate for control of the Nexus auxiliary port when Nexus data is ready to be transmitted.

All modules arbitrating for the port are given fixed priority levels relative to each other. If multiple modules have the same request level, this priority level is used as a tie-breaker. To avoid monopolization of the port, the module given the highest priority level alternates following each grant. Immediately out of reset the order of priority, from highest to lowest, is: NPC, Nexus3+, and Nexus2+. This arbitration mechanism is controlled internally and is not programmable by tools or the user.

36.5.5.2.5 Nexus JTAG Port Sharing

Each of the individual Nexus modules on the device implements a TAP controller for accessing its registers. When JCOMP is asserted, only the module whose ACCESS_AUX_TAP instruction is loaded

has control of the TAP (see Section 35.4.4, JTAGC Instructions). This allows the interface to all of these individual TAP controllers to appear to be a single port from outside the device. Once a Nexus module has ownership of the TAP, that module acts like a single-bit shift register, or bypass register, if no register is selected as the shift path.

36.5.5.2.6 MCKO

MCKO is an output clock to the development tools used for the timing of $\overline{\text{MSEO}}$ and MDO pin functions. MCKO is derived from the system clock and its frequency is determined by the value of the MCKO_DIV[2:0] field in the PCR. Possible operating frequencies include full, one-half, one-quarter, and one-eighth SYS_CLK speed. MCKO is enabled by setting the MCKO_EN bit in the PCR.

The NPC also controls dynamic MCKO clock gating. The setting of the MCKO_GT bit inside the PCR determines whether or not MCKO gating control is enabled. The MCKO_GT bit resets to a logic 0. In this state gating of MCKO is disabled. To enable gating of MCKO, the MCKO_GT bit in the PCR is written to a logic 1. When MCKO gating is enabled, MCKO is driven to a logic 0 if the auxiliary port is enabled but not transmitting messages and there are no pending messages from Nexus clients.

36.5.5.2.7 $\overline{\text{EVTO}}$ Sharing

The NPC controls sharing of the $\overline{\text{EVTO}}$ output between all Nexus clients that produce an $\overline{\text{EVTO}}$ signal. $\overline{\text{EVTO}}$ is driven for one MCKO period whenever any module drives its $\overline{\text{EVTO}}$. The sharing mechanism is a logical AND of all incoming $\overline{\text{EVTO}}$ signals from Nexus blocks, thereby asserting $\overline{\text{EVTO}}$ whenever any block drives its $\overline{\text{EVTO}}$. The order these signals are connected at the NPC input does not matter. When no MCKO is active, such as in disabled mode, the NPC assumes an MCKO frequency of one-half system clock speed when driving $\overline{\text{EVTO}}$. $\overline{\text{EVTO}}$ sharing is active as long as the NPC is not in reset.

36.5.5.2.8 Nexus Reset Control

The JCOMP input that is used as the primary reset signal for the NPC is also used by the NPC to generate a single-bit reset signal for other Nexus blocks. If JCOMP is negated, an internal reset signal is asserted, indicating that all Nexus modules should be held in reset. This internal reset signal is also asserted during a power-on-reset. The single bit reset signal functions much like the IEEE 1149.1-2001 defined $\overline{\text{TRST}}$ signal and allows JCOMP reset information to be provided to the Nexus blocks without each block having to sense the JCOMP signal directly.

36.5.5.2.9 Processor Status (PSTAT) Muxing

PSTAT mode is intended for factory processor debug only.

36.5.6 NPC Initialization/Application Information

To initialize the TAP for NPC register accesses, the following sequence is required:

1. Enable the NPC TAP controller. This is achieved by asserting JCOMP and loading the ACCESS_AUX_TAP_NPC instruction in the JTAGC.
2. Load the TAP controller with the NEXUS-ENABLE instruction.

To write control data to NPC tool-mapped registers, the following sequence is required:

1. Write the 7-bit register index and set the write bit to select the register with a pass through the SELECT-DR-SCAN path in the TAP controller state machine.
2. Write the register value with a second pass through the SELECT-DR-SCAN path. Note that the prior value of this register is shifted out during the write.

To read status and control data from NPC tool-mapped registers, the following sequence is required:

1. Write the 7-bit register index and clear the write bit to select register with a pass through SELECT-DR-SCAN path in the TAP controller state machine.
2. Read the register value with a second pass through the SELECT-DR-SCAN path. Data shifted in is ignored.

See the IEEE®-ISTO 5001-2003 standard for more detail.

36.6 e200z6 Class 3 Nexus Module (Nexus3+)

The Nexus3+ module provides real-time development capabilities for the device core in compliance with the IEEE®-ISTO Nexus 5001-2003 standard. This module provides development support capabilities without requiring the use of address and data pins for internal visibility.

A portion of the pin interface (the JTAG port) is also shared with the OnCE / Nexus 1 unit. The **IEEE-ISTO 5001-2003** standard defines an extensible auxiliary port which is used in conjunction with the JTAG port in e200z6 processors.

36.6.1 Nexus3+ Introduction

This section defines the auxiliary pin functions, transfer protocols and standard development features of the Nexus3+ module. The development features supported are Program Trace, Data Trace, Watchpoint Messaging, Ownership Trace, and Read/Write access via the JTAG interface. The Nexus3+ module also supports two Class 4 features: Watchpoint Triggering and Processor Overrun Control.

NOTE

Throughout this section references are made to the auxiliary port and its specific signals, such as MCKO, $\overline{\text{MSEO}}$[1:0], MDO[11:0] and others. In actual use, the device NPC module arbitrates the access of the single auxiliary port. To simplify the description of the function of the Nexus3+ module, the interaction of the NPC is omitted and the behavior described as if the module has its own dedicated auxiliary port.

36.6.2 Nexus3+ Block Diagram

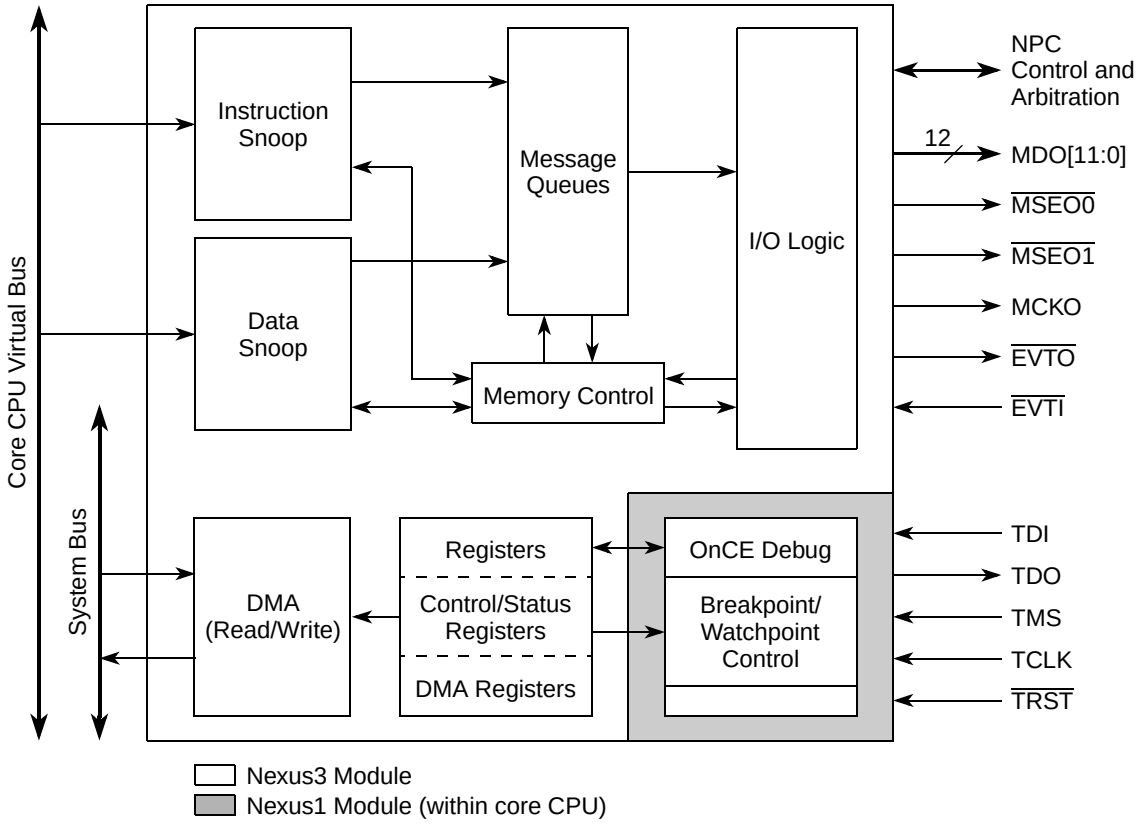


Figure 36-13. e200z6 Nexus3 Functional Block Diagram

36.6.3 Nexus3+ Overview

Table 36-13 contains a set of terms and definitions associated with the Nexus3+ module.

Table 36-13. Terms and Definitions

Term	Description
IEEE®-ISTO 5001	Consortium and standard for real-time embedded system design. World wide Web documentation at http://www.ieee-isto.org/Nexus5001
Auxiliary Port	Refers to Nexus auxiliary port. Used as auxiliary port to the IEEE® 1149.1 JTAG interface.
Branch Trace Messaging (BTM)	Visibility of addresses for taken branches and exceptions, and the number of sequential instructions executed between each taken branch.
Client	A functional block on an embedded processor that requires development visibility and controllability. Examples are a central processing unit (CPU) or an intelligent peripheral.
Data Read Message (DRM)	External visibility of data reads to memory-mapped resources.
Data Write Message (DWM)	External visibility of data writes to memory-mapped resources.
Data Trace Messaging (DTM)	External visibility of how data flows through the embedded system. This may include DRM and/or DWM.

Table 36-13. Terms and Definitions (continued)

Term	Description
JTAG Compliant	Device complying to IEEE® 1149.1 JTAG standard
JTAG IR & DR Sequence	JTAG instruction register (IR) scan to load an opcode value for selecting a development register. The JTAG IR corresponds to the OnCE command register (OCMD). The selected development register is then accessed via a JTAG data register (DR) scan.
Nexus1	The e200z6 (OnCE) debug module. This module integrated with each e200z6 processor provides all static (core halted) debug functionality. This module is compliant with Class1 of the IEEE®-ISTO 5001 standard.
Ownership Trace Message (OTM)	Visibility of process/function that is currently executing.
Public Messages	Messages on the auxiliary pins for accomplishing common visibility and controllability requirements
Standard	The phrase ‘according to the standard’ is used to indicate according to the IEEE®-ISTO 5001 standard.
Transfer Code (TCODE)	Message header that identifies the number and/or size of packets to be transferred, and how to interpret each of the packets.
Watchpoint	A data or instruction breakpoint that does not cause the processor to halt. Instead, a pin is used to signal that the condition occurred. A watchpoint message is also generated.

36.6.4 Nexus3+ Features

The Nexus3+ module is compliant with Class 3 of the IEEE®-ISTO 5001-2003 standard. The following features are implemented:

- Program trace via branch trace messaging (BTM). Branch trace messaging displays program flow discontinuities (direct and indirect branches, exceptions, etc.), allowing the development tool to interpolate what transpires between the discontinuities. Thus static code may be traced.
- Data trace via data write messaging (DWM) and data read messaging (DRM). This provides the capability for the development tool to trace reads and/or writes to selected internal memory resources.
- Ownership trace via ownership trace messaging (OTM). OTM facilitates ownership trace by providing visibility of which process ID or operating system task is activated. An ownership trace message is transmitted when a new process/task is activated, allowing the development tool to trace ownership flow.
- Run-time access to embedded processor registers and memory map via the JTAG port. This allows for enhanced download/upload capabilities.
- Watchpoint messaging via the auxiliary pins.
- Watchpoint trigger enable of Program and/or Data Trace Messaging.
- Higher speed data input/output via the auxiliary port.
- Registers for Program Trace, Data Trace, Ownership Trace and Watchpoint Trigger.
- All features controllable and configurable via the JTAG port.

36.6.5 Enabling Nexus3+ Operation

The Nexus3+ module is enabled by loading a single instruction (ACCESS_AUX_TAP_ONCE, as shown in Table 35-2) into the JTAGC instruction register (IR), and then loading the corresponding OnCE OCMD register with the NEXUS3_ACCESS instruction (refer to Table 36-2). For the e200z6 Class 3+ Nexus module, the OCMD value is 0b00_0111_1100. Once enabled, the module is ready to accept control input via the JTAG pins. See Section 36.4.1.1, Enabling Nexus Clients for TAP Access for more information.

Enabling the Nexus3+ module automatically enables the generation of Debug Status Messages.

The Nexus module is disabled when the JTAG state machine reaches the test-logic-reset state. This state can be reached by the negation of the JCOMP pin or by cycling through the state machine using the TMS pin. The Nexus module also is disabled if a power-on-reset (POR) event occurs. If the Nexus3+ module is disabled, no trace output is provided, and the module disables (drives inactive) auxiliary port output pins MDO[11:0], $\overline{\text{MSEO}}$[1:0], and MCKO. Nexus registers are not available for reads or writes.

36.6.6 TCODEs Supported by Nexus3+

The Nexus3+ pins allow for flexible transfer operations via public messages. A TCODE defines the transfer format, the number and/or size of the packets to be transferred, and the purpose of each packet. The IEEE-ISTO 5001-2003 standard defines a set of public messages. The Nexus3+ module supports the public TCODEs seen in Table 36-14. Each message contains multiple packets transmitted in the order shown in the table.

Table 36-14. Public TCODEs Supported by Nexus3+

Message Name	Packet Size (bits)		Packet Name	Packet Type	Packet Description
	Min	Max			
Debug Status	6	6	TCODE	Fixed	TCODE number = 0 (0x00)
	4	4	SRC	Fixed	source processor identifier
	8	8	STATUS	Fixed	Debug status register (DS[31:24])
Ownership Trace Message	6	6	TCODE	Fixed	TCODE number = 2 (0x02)
	4	4	SRC	Fixed	source processor identifier
	32	32	PROCESS	Fixed	Task/Process ID tag
Program Trace — Direct Branch Message[1]	6	6	TCODE	Fixed	TCODE number = 3 (0x03)
	4	4	SRC	Fixed	source processor identifier
	1	8	I-CNT	Variable	# sequential instructions executed since last taken branch
Program Trace — Indirect Branch Message[1]	6	6	TCODE	Fixed	TCODE number = 4 (0x04)
	4	4	SRC	Fixed	source processor identifier
	1	8	I-CNT	Variable	# sequential instructions executed since last taken branch
	1	32	U-ADDR	Variable	unique part of target address for taken branches/exceptions

Table 36-14. Public TCODEs Supported by Nexus3+ (continued)

Message Name	Packet Size (bits)		Packet Name	Packet Type	Packet Description
	Min	Max			
Data Trace — Data Write Message	6	6	TCODE	Fixed	TCODE number = 5 (0x05)
	4	4	SRC	Fixed	source processor identifier
	3	3	DSIZ	Fixed	data size (Refer to Table 36-18)
	1	32	U-ADDR	Variable	unique portion of the data write address
	1	64	DATA	Variable	data write values (see Section 36.6.10.4, Data Trace, for details)
Data Trace — Data Read Message	6	6	TCODE	Fixed	TCODE number = 6 (0x06)
	4	4	SRC	Fixed	source processor identifier
	3	3	DSIZ	Fixed	data size (Refer to Table 36-18)
	1	32	U-ADDR	Variable	unique portion of the data read address
	1	64	DATA	Variable	data read values (see Section 36.6.10.4, Data Trace, for details)
Error Message	6	6	TCODE	Fixed	TCODE number = 8 (0x08)
	4	4	SRC	Fixed	source processor identifier
	5	5	ECODE	Fixed	error code
Program Trace — Direct Branch Message w/ Sync[1]	6	6	TCODE	Fixed	TCODE number = 11 (0x0B)
	4	4	SRC	Fixed	source processor identifier
	1	8	I-CNT	Variable	# sequential instructions executed since last taken branch
	1	32	F-ADDR	Variable	full target address (leading zeros truncated)
Program Trace — Indirect Branch Message w/ Sync[1]	6	6	TCODE	Fixed	TCODE number = 12 (0x0C)
	4	4	SRC	Fixed	source processor identifier
	1	8	I-CNT	Variable	# sequential instructions executed since last taken branch
	1	32	F-ADDR	Variable	full target address (leading zeros truncated)
Data Trace — Data Write Message w/ Sync	6	6	TCODE	Fixed	TCODE number = 13 (0x0D)
	4	4	SRC	Fixed	source processor identifier
	3	3	DSZ	Fixed	data size (Refer to Table 36-18)
	1	32	F-ADDR	Variable	full access address (leading zeros truncated)
	1	64	DATA	Variable	data write values (see Section 36.6.10.4, Data Trace, for details)

Table 36-14. Public TCODEs Supported by Nexus3+ (continued)

Message Name	Packet Size (bits)		Packet Name	Packet Type	Packet Description
	Min	Max			
Data Trace — Data Read Message w/ Sync	6	6	TCODE	Fixed	TCODE number = 14 (0x0E)
	4	4	SRC	Fixed	source processor identifier
	3	3	DSZ	Fixed	data size (Refer to Table 36-18)
	1	32	F-ADDR	Variable	full access address (leading zeros truncated)
	1	64	DATA	Variable	data read values (see Section 36.6.10.4, Data Trace, for details)
Watchpoint Message	6	6	TCODE	Fixed	TCODE number = 15 (0x0F)
	4	4	SRC	Fixed	source processor identifier
	4	4	WPHIT	Fixed	# indicating watchpoint sources
Resource Full Message	6	6	TCODE	Fixed	TCODE number = 27 (0x1B)
	4	4	SRC	Fixed	source processor identifier
	4	4	RCODE	Fixed	resource code (Refer to RCODE values in Table 36-17) - indicates which resource is the cause of this message
	1	32	RDATA	Variable	branch / predicate instruction history (see Section 36.6.10.3.1, Branch Trace Messaging (BTM))
Program Trace — Indirect Branch History Message	6	6	TCODE	Fixed	TCODE number = 28 (0x1C) (see footnote 1 below)
	4	4	SRC	Fixed	source processor identifier
	1	8	I-CNT	Variable	# sequential instructions executed since last taken branch
	1	32	U-ADDR	Variable	unique part of target address for taken branches/exceptions
	1	32	HIST	Variable	branch / predicate instruction history (see Section 36.6.10.3.1, Branch Trace Messaging (BTM))
Program Trace — Indirect Branch History Message w/ Sync	6	6	TCODE	Fixed	TCODE number = 29 (0x1D) (see footnote 1 below)
	4	4	SRC	Fixed	source processor identifier
	1	8	I-CNT	Variable	# sequential instructions executed since last taken branch
	1	32	F-ADDR	Variable	full target address (leading zero (0) truncated)
	1	32	HIST	Variable	branch / predicate instruction history (see Section 36.6.10.3.1, Branch Trace Messaging (BTM))
Program Trace — Program Correlation Message	6	6	TCODE	Fixed	TCODE number = 33 (0x21)
	4	4	SRC	Fixed	source processor identifier
	4	4	EVCODE	Fixed	event correlated w/ program flow (Refer to Table 36-17)
	1	8	I-CNT	Variable	# sequential instructions executed since last taken branch
	1	32	HIST	Variable	branch / predicate instruction history (see Section 36.6.10.3.1, Branch Trace Messaging (BTM))

[1] The user can select between the two types of program trace. The advantages for each are discussed in Section 36.6.10.3.1, Branch Trace Messaging (BTM). If the branch history method is selected, the shaded TCODEs are not messaged out.

Table 36-15 shows the error code encodings used when reporting an error via the Nexus3 Error Message.

Table 36-15. Error Code Encoding (TCODE = 8)

Error Code (ECODE)	Description
00000	Ownership trace overrun
00001	Program trace overrun
00010	Data trace overrun
00011	Read/write access error
00101	Invalid access opcode (Nexus register unimplemented)
00110	Watchpoint overrun
00111	(Program Trace or Data Trace) and Ownership Trace overrun
01000	(Program Trace or Data Trace or Ownership Trace) and Watchpoint overrun
01001–0111	Reserved
11000	BTM lost due to collision w/ higher priority message
11001–11111	Reserved

Table 36-16 shows the encodings used for resource codes for certain messages.

Table 36-16. RCODE values (TCODE = 27)

Resource Code (RCODE)	Description	Resource Data (RDATA)
0000	Program Trace Instruction Counter overflow (reached 255 and was reset)	0xFF
0001	Program Trace, Branch / Predicate Instruction History. This type of packet is terminated by a stop bit set to 1 after the last history bit.	Branch History. This type of packet is terminated by a stop bit set to a 1 after the last history bit.

Table 36-17 shows the event code encodings used for certain messages.

Table 36-17. Event Code Encoding (TCODE = 33)

Event Code	Description
0000	Entry into Debug Mode
0001	Entry into Low Power Mode (CPU only)[1]
0010–0011	Reserved for future functionality
0100	Disabling Program Trace
0101–1101	Reserved for future functionality

Table 36-17. Event Code Encoding (TCODE = 33)

Event Code	Description
1110	Entry into a VLE page from a non-VLE page
1111	Entry into a non-VLE page from a VLE page

1 The device enters Low Power Mode when the Nexus stall mode is enabled (Nexus3_DC1[OVC] = 0b011) and a trace message is in danger of over-flowing the Nexus queue.

Table 36-18 shows the data trace size encodings used for certain messages.

Table 36-18. Data Trace Size Encodings (TCODE = 5, 6, 13, 14)

DTM Size Encoding	Transfer Size
000	Byte
001	Half-word (2 bytes)
010	Word (4 bytes)
011	Double-word (8 bytes)
100	String (3 bytes)
101–111	Reserved

36.6.7 Nexus3+ Memory Map

This section describes the Nexus3+ programmer's model. Nexus3+ registers are accessed using the JTAG/OnCE port in compliance with IEEE® 1149.1. See Section 36.6.9, Nexus3+ Register Access via JTAG / OnCE for details on Nexus3+ register access.

NOTE

Nexus3+ registers and output signals are numbered using bit 0 as the least significant bit. This bit ordering is consistent with the ordering defined by the IEEE®-ISTO 5001 standard.

Table 36-19 details the register map for the Nexus3+ module.

Table 36-19. Nexus3+ Memory Map

Access Opcode	Register Name	Register Description	Read Address	Write Address
0x2	DC1	Development control 1	0x04	0x05
0x3	DC2	Development control 2	0x06	0x07
0x4	DS	Development status	0x08	—
0x7	RWCS	Read/write access control/status	0x0E	0x0F
0x9	RWA	Read/write access address	0x12	0x13
0xA	RWD	Read/write access data	0x14	0x15

Table 36-19. Nexus3+ Memory Map (continued)

Access Opcode	Register Name	Register Description	Read Address	Write Address
0xB	WT	Watchpoint trigger	0x16	0x17
0xD	DTC	Data trace control	0x1A	0x1B
0xE	DTSA1	Data trace start address 1	0x1C	0x1D
0xF	DTSA2	Data trace start address 2	0x1E	0x1F
0x12	DTEA1	Data trace end address 1	0x24	0x25
0x13	DTEA2	Data trace end address 2	0x26	0x27
0x14 -> 0x3F	—	Reserved	0x28->0x7E	0x29->0x7F

36.6.8 Nexus3+ Register Definition

36.6.8.1 Development Control Register 1, 2 (DC1, DC2)

The development control registers are used to control the basic development features of the Nexus module. Figure 36-14 shows DC1 and Table 36-20 describes the register's fields.

Nexus Reg: 0x02 Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	OPC	MCK_DIV		EOC		0	PTM	WEN	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	OVC			EIC		TM		
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 36-14. Development Control Register 1 (DC1)

Table 36-20. DC1 Field Descriptions

Field	Description
OPC[1]	Output Port Mode Control. 0 Reduced-port mode configuration (4 MDO pins). 1 Full-port mode configuration (8 MDO pins).
MCK_DIV[1]	MCKO Clock Divide Ratio (see note below). 00 MCKO is 1x processor clock freq. 01 MCKO is 1/2x processor clock freq. 10 MCKO is 1/4x processor clock freq. 11 MCKO is 1/8x processor clock freq.

Table 36-20. DC1 Field Descriptions (continued)

Field	Description
EOC	$\overline{\text{EVTO}}$ Control. 00 $\overline{\text{EVTO}}$ upon occurrence of watchpoints (configured in DC2). 01 $\overline{\text{EVTO}}$ upon entry into debug mode. 10 $\overline{\text{EVTO}}$ upon timestamping event. 11 Reserved.
PTM	Program Trace Method. 0 Program trace uses traditional branch messages. 1 Program trace uses branch history messages.
WEN	Watchpoint Trace Enable. 0 Watchpoint messaging disabled. 1 Watchpoint messaging enabled.
OVC	Overrun Control. 000 Generate overrun messages. 001–010 Reserved. 011 Delay processor for BTM / DTM / OTM overruns. 1XX Reserved.
EIC	$\overline{\text{EVTI}}$ Control. 00 $\overline{\text{EVTI}}$ is used for synchronization (program trace/ data trace). 01 $\overline{\text{EVTI}}$ is used for debug request. 1X Reserved.
TM	Trace Mode. Any or all of the TM bits may set, enabling one or more traces. 000 No trace. 1XX Program trace enabled. X1X Data trace enabled. XX1 Ownership trace enabled.

[1] The output port mode control bit (OPC) and MCKO divide bits (MCK_DIV) are shown for clarity. These functions are controlled globally by the NPC port control register (PCR). These bits are writable in the PCR but have no effect.

DC2 is shown in Figure 36-15 and its fields are described in Table 36-21.

Nexus Reg: 0x3 Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	EWC								0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 36-15. Development Control Register 2 (DC2)

Table 36-21. DC2 Field Descriptions

Field	Description
EWC	$\overline{\text{EVTO}}$ Watchpoint Configuration. Any or all of the bits in EWC may be set to configure the $\overline{\text{EVTO}}$ watchpoint. 0000_0000 No Watchpoints trigger $\overline{\text{EVTO}}$ 1xxx_xxxx Watchpoint #0 (IAC1 from Nexus1) triggers $\overline{\text{EVTO}}$. x1xx_xxxx Watchpoint #1 (IAC2 from Nexus1) triggers $\overline{\text{EVTO}}$. xx1x_xxxx Watchpoint #2 (IAC3 from Nexus1) triggers $\overline{\text{EVTO}}$. xxx1_xxxx Watchpoint #3 (IAC4 from Nexus1) triggers $\overline{\text{EVTO}}$. xxxx_1xxx Watchpoint #4 (DAC1 from Nexus1) triggers $\overline{\text{EVTO}}$. xxxx_x1xx Watchpoint #5 (DAC2 from Nexus1) triggers $\overline{\text{EVTO}}$. xxxx_xx1x Watchpoint #6 (DCNT1 from Nexus1) triggers $\overline{\text{EVTO}}$. xxxx__xxx1 Watchpoint #7 (DCNT2 from Nexus1) triggers $\overline{\text{EVTO}}$.

NOTE

The EOC bits in DC1 must be programmed to trigger $\overline{\text{EVTO}}$ on watchpoint occurrence for the EWC bits to have any effect.

36.6.8.2 Development Status Register (DS)

The development status register is used to report system debug status. When debug mode is entered or exited, or an e200z6-defined low-power mode is entered, a debug status message is transmitted with DS[31:24]. The external tool can read this register at any time. The DS register is shown in Figure 36-16 and its fields are described in Table 36-22.

Nexus Reg: 0x4 Access: User read only

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	DBG	0	0	0	LPC		CHK	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 36-16. Development Status Register (DS)

Table 36-22. DS Field Descriptions

Field	Description
DBG	CPU Debug Mode Status. 0 CPU not in debug mode. 1 CPU in debug mode.

Table 36-22. DS Field Descriptions (continued)

Field	Description
LPC	CPU Low-Power Mode Status. 00 Normal (run) mode. 01 CPU in halted state. 10 CPU in stopped state. 11 Reserved.
CHK	CPU Checkstop Status. 0 CPU not in checkstop state. 1 CPU in checkstop state.

36.6.8.3 Read/Write Access Control/Status (RWCS)

The read write access control/status register provides control for read/write access. Read/write access provides DMA-like access to memory-mapped resources on the system bus while the processor is halted or during runtime. The RWCS register is shown in Figure 36-17 and its fields are described in Table 36-23. The RWCS register also provides read/write access status information as shown in Table 36-24.

Nexus Reg: 0x7 Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	AC	RW	SZ			MAP			PR		BST	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	CNT														ERR	DV
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 36-17. Read/Write Access Control/Status Register (RWCS)

Table 36-23. RWCS Field Description

Field	Description
AC	Access Control. 0 End access. 1 Start access.
RW	Read/Write Select. 0 Read access. 1 Write access.
SZ	Word Size. 000 8-bit (byte.) 001 16-bit (halfword). 010 32-bit (word). 011 64-bit (doubleword—only in burst mode). 100–111 Reserved (default to word).

Table 36-23. RWCS Field Description (continued)

Field	Description
MAP	MAP Select. 000 Primary memory map. 001–111 Reserved.
PR	Read/Write Access Priority. 00 Lowest access priority. 01 Reserved (default to lowest priority). 10 Reserved (default to lowest priority). 11 Highest access priority.
BST	Burst Control. 0 Module accesses are single bus cycle at a time. 1 Module accesses are performed as burst operation.
CNT	Access Control Count. Number of accesses of word size SZ.
ERR	Read/Write Access Error. See Table 36-24.
DV	Read/Write Access Data Valid. See Table 36-24.

Table 36-24 details the status bit encodings.

Table 36-24. Read/Write Access Status Bit Encoding

Read Action	Write Action	ERR	DV
Read access has not completed	Write access completed without error	0	0
Read access error has occurred	Write access error has occurred	1	0
Read access completed without error	Write access has not completed	0	1
Not allowed	Not allowed	1	1

36.6.8.4 Read/Write Access Address (RWA)

The read/write access address register provides the system bus address to be accessed when initiating a read or a write access.

Nexus Reg: 0x9 Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	Read/Write Address																															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 36-18. Read/Write Access Address Register (RWA)

36.6.8.5 Read/Write Access Data (RWD)

The read/write access data register provides the data to/from system bus memory-mapped locations when initiating a read or a write access.

Nexus Reg: 0xA — Access: User read/write

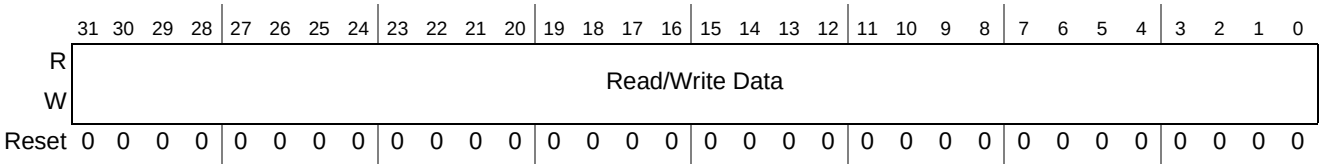


Figure 36-19. Read/Write Access Data Register (RWD)

Table 36-25 shows the proper placement of data into the RWD. The “X” in the RWD column indicate byte lanes with valid data.

Table 36-25. RWD Data Placement for Transfers

Transfer Size and byte offset	RWA[2:0	RWCS[SZ]	RWD			
			31:24	23:16	15:8	7:0
Byte	*xxx*	000	—	—	—	X
Half Word	*xx*0	001	—	—	X	X
Word	*x*00	010	X	X	X	X
Double Word	000	011				
first RWD pass (low order data)			X	X	X	X
second RWD pass (high order data)			X	X	X	X

Table 36-26 shows the mapping of RWD bytes to byte lanes of the AHB read and write data buses.

Table 36-26. RWD data placement for Transfers

Transfer Size and byte offset	RWA[2:0]	RWD			
		31:24	23:16	15:8	7:0
Byte @000	000	—	—	—	AHB[7:0]
Byte @001	001	—	—	—	AHB[15:8]
Byte @010	010	—	—	—	AHB[23:16]
Byte @011	011	—	—	—	AHB[31:24]
Byte @100	100	—	—	—	AHB[39:32]
Byte @101	101	—	—	—	AHB[[47:40]
Byte @110	110	—	—	—	AHB[55:48]
Byte @111	111	—	—	—	AHB[63:56]
Half@000	000	—	—	AHB[15:8]	AHB[7:0]
Half@010	010	—	—	AHB[31:24]	AHB[23:16]
Half@100	100	—	—	AHB[[47:40]	AHB[39:32]
Half@110	110	—	—	AHB[63:56]	AHB[55:48]
Word@000	000	AHB[31:24]	AHB[23:16]	AHB[15:8]	AHB[7:0]

Table 36-26. RWD data placement for Transfers

Transfer Size and byte offset	RWA[2:0]	RWD			
		31:24	23:16	15:8	7:0
Word@100	100	AHB[63:56]	AHB[55:48]	AHB[[47:40]	AHB[39:32]
Doubleword@000	000				
first RWD pass		AHB[31:24]	AHB[23:16]	AHB[15:8]	AHB[7:0]
second RWD pass		AHB[63:56]	AHB[55:48]	AHB[[47:40]	AHB[39:32]

NOTE

The Nexus/JTAG Read/Write Access Control/Status Register (RWCS) write (to begin a read access) or the write to the Read/Write Access Data Register (RWD)(to begin a write access) does not actually begin its action until one JTAG clock (TCK) after leaving the JTAG Update-DR state. This prevents the access from being performed and therefore will not signal its completion via the READY (RDY) output unless the JTAG controller receives an additional TCK. In addition, EVTI is not latched into the device unless there are clock transitions on TCK.

Therefore, the tool/debugger must provide at least one TCK clock for the EVTI signal to be recognized by the MCU. When using the RDY signal to indicate the end of a Nexus read/write access, ensure that TCK continues to run for at least one TCK after leaving the Update-DR state. This can be just a TCK with TMS low while in the Run-Test/Idle state or by continuing with the next Nexus/JTAG command. Expect the affect of EVTI and RDY to be delayed by edges of TCK. RDY is not available in all packages of all devices.

36.6.8.6 Watchpoint Trigger Register (WT)

The watchpoint trigger register allows the watchpoints defined within the e200z6 Nexus1 logic to trigger actions. These watchpoints can control program and/or data trace enable and disable. The WT bits can be used to produce an address-related window for triggering trace messages. The WT register is shown in Figure 36-20 and its fields are described in Table 36-27.

Nexus Reg: 0xB

Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	PTS				PTS				DTS				DTE				0	0	0	0
W																				
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 36-20. Watchpoint Trigger Register (WT)

Table 36-27. WT Field Descriptions

Field	Description
PTS	Program Trace Start Control. 000 Trigger disabled. 001 Use watchpoint #0 (IAC1 from Nexus1). 010 Use watchpoint #1 (IAC2 from Nexus1). 011 Use watchpoint #2 (IAC3 from Nexus1). 100 Use watchpoint #3 (IAC4 from Nexus1). 101 Use watchpoint #4 (DAC1 from Nexus1). 110 Use watchpoint #5 (DAC2 from Nexus1). 111 Use watchpoint #6 or #7 (DCNT1 or DCNT2 from Nexus1).
PTE	Program Trace End Control. 000 Trigger disabled. 001 Use watchpoint #0 (IAC1 from Nexus1). 010 Use watchpoint #1 (IAC2 from Nexus1). 011 Use watchpoint #2 (IAC3 from Nexus1). 100 Use watchpoint #3 (IAC4 from Nexus1). 101 Use watchpoint #4 (DAC1 from Nexus1). 110 Use watchpoint #5 (DAC2 from Nexus1). 111 Use watchpoint #6 or #7 (DCNT1 or DCNT2 from Nexus1).

Table 36-27. WT Field Descriptions (continued)

Field	Description
DTS	Data Trace Start Control. 000 Trigger disabled. 001 Use watchpoint #0 (IAC1 from Nexus1). 010 Use watchpoint #1 (IAC2 from Nexus1). 011 Use watchpoint #2 (IAC3 from Nexus1). 100 Use watchpoint #3 (IAC4 from Nexus1). 101 Use watchpoint #4 (DAC1 from Nexus1). 110 Use watchpoint #5 (DAC2 from Nexus1). 111 Use watchpoint #6 or #7 (DCNT1 or DCNT2 from Nexus1).
DTE	Data Trace End Control. 000 Trigger disabled. 001 Use watchpoint #0 (IAC1 from Nexus1). 010 Use watchpoint #1 (IAC2 from Nexus1). 011 Use watchpoint #2 (IAC3 from Nexus1). 100 Use watchpoint #3 (IAC4 from Nexus1). 101 Use watchpoint #4 (DAC1 from Nexus1). 110 Use watchpoint #5 (DAC2 from Nexus1). 111 Use watchpoint #6 or #7 (DCNT1 or DCNT2 from Nexus1).

NOTE

The WT bits can control program/data trace only if the TM bits in the development control register 1 (DC1) have not already been set to enable program and data trace, respectively.

36.6.8.7 Data Trace Control Register (DTC)

The data trace control register controls whether DTM messages are restricted to reads, writes, or both for a user programmable address range. Two data trace channels are controlled by the DTC for the Nexus3 module. Each channel can also be programmed to trace data accesses or instruction accesses.

Nexus Reg: 0xD — Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	RWT1		RWT2		0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	RC1	RC2	0	0	DI1	DI2	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 36-21. Data Trace Control Register (DTC)

Table 36-28 details the data trace control register fields.

Table 36-28. DTC Field Description

Field	Description
31–30 RWT1[1:0]	Read/write trace 1. 00 No trace enabled. *x*1 Enable data read trace. 1*x* Enable data write trace.
29–28 RWT2[1:0]	Read/write trace 2. 00 No trace enabled. *x*1 Enable data read trace. 1*x* Enable data write trace.
7 RC1	Range control 1. 0 Condition trace on address within range. 1 Condition trace on address outside of range.
6 RC2	Range control 2. 0 Condition trace on address within range. 1 Condition trace on address outside of range.
3 DI1	Data access/instruction access trace 1. 0 Condition trace on data accesses. 1 Condition trace on instruction accesses.
2 DI2	Data access/instruction access trace 2. 0 Condition trace on data accesses. 1 Condition trace on instruction accesses.

36.6.8.8 Data Trace Start Address Registers 1 and 2 (DTSA*n*)

The data trace start address registers define the start addresses for each trace channel.

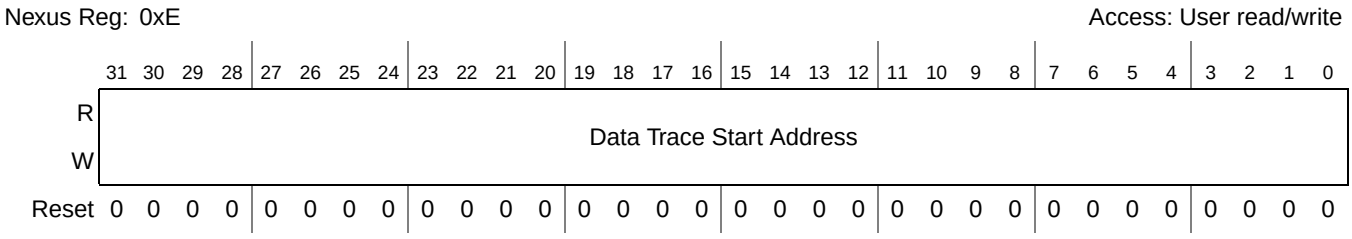


Figure 36-22. Data Trace Start Address Register 1 (DTSA1)

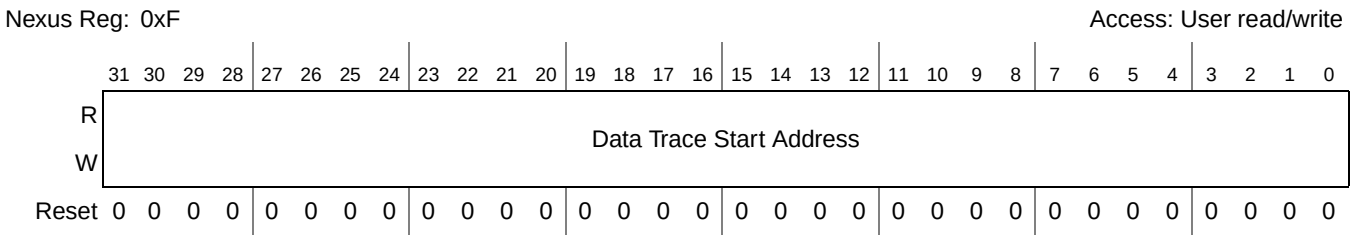


Figure 36-23. Data Trace Start Address Register 2 (DTSA2)

36.6.8.9 Data Trace End Address Registers 1 and 2 (DTEA*n*)

The data trace end address registers define the end addresses for each trace channel.

Nexus Reg: 0x12 Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	Data Trace End Address																															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 36-24. Data Trace End Address Register 1 (DTEA1)

Nexus Reg: 0x13 Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	Data Trace End Address																															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 36-25. Data Trace End Address Register 2 (DTEA2)

Table 36-29 illustrates the range that selected for data trace for various cases of DTSA being less than, greater than, or equal to DTEA.

Table 36-29. Data Trace—Address Range Options

Programmed Values	Range Control Bit Value	Range Selected
DTSA < DTEA	0	DTSA → ←DTEA
DTSA < DTEA	1	← DTSA DTEA →
DTSA > DTEA	N/A	Invalid range—no trace
DTSA = DTEA	N/A	Invalid range—no trace

NOTE

DTSA must be less than DTEA in order to guarantee correct data write/read traces. Data trace ranges are inclusive of the DTSA and DTEA addresses for Range Control settings indicating “within range”, and are exclusive of the DTSA and DTEA addresses for Range Control settings indicating “outside of range.”

36.6.9 Nexus3+ Register Access via JTAG / OnCE

Access to Nexus3 register resources is enabled by loading a single instruction (ACCESS_AUX_TAP_Z6) into the JTAGC instruction register (IR), and then loading the corresponding OnCE OCMD register with the NEXUS3_ACCESS instruction (refer to Table 36-2). For the Nexus3+ module, the OCMD value is 0b00_0111_1100.

Once the ACCESS_AUX_TAP_Z6 instruction has been loaded, the JTAG/OnCE port allows tool/target communications with all Nexus3 registers according to the register map in Table 36-19.

Reading/writing of a Nexus3+ register then requires two (2) passes through the data-scan (DR) path of the JTAG state machine (see 36.6.10.8).

1. The first pass through the DR selects the Nexus3+ register to be accessed by providing an index (see Table 36-19), and the direction (read/write). This is achieved by loading an 8-bit value into the JTAG data register (DR). This register has the following format:

(7 bits)	(1 bit)
Nexus Register Index	R/W

RESET Value: 0x00

Nexus Register Index:	Selected from values in Table 36-19
Read/Write (R/W)	0 Read 1 Write

2. The second pass through the DR then shifts the data in or out of the JTAG port, LSB first.
   a) During a read access, data is latched from the selected Nexus register when the JTAG state machine passes through the CAPTURE-DR state.
   b) During a write access, data is latched into the selected Nexus register when the JTAG state machine passes through the UPDATE-DR state.

36.6.10 Nexus3+ Functional Description

36.6.10.1 Debug Status Messages

Debug Status Messages report low power mode and debug status. Debug Status Messages are enabled when Nexus 3+ is enabled. Entering/exiting Debug Mode as well as entering a Low Power Mode triggers a Debug Status Message, indicating the value of the most significant byte in the Development Status register. Debug status information is sent out in the following format:

Table 36-30. Debug Status Message Format

(8 bits)	(4 bits)	(6 bits)
DS[31:24]	Source Processor	TCODE (000000)

Fixed Length = 18 bits

36.6.10.2 Ownership Trace

This section details the ownership trace features of the Nexus3+ module.

36.6.10.2.1 Overview

Ownership trace provides a macroscopic view, such as task flow reconstruction, when debugging software written in a high level (or object-oriented) language. It offers the highest level of abstraction for tracking operating system software execution. This is especially useful when the developer is not interested in debugging at lower levels.

36.6.10.2.2 Ownership Trace Messaging (OTM)

Ownership trace information is messaged via the auxiliary port using an ownership trace message (OTM). The e200z6 processor contains a Power Architecture Book E defined process ID register within the CPU.

The process ID register is updated by the operating system software to provide task/process ID information. The contents of this register are replicated on the pins of the processor and connected to Nexus. The process ID register value can be accessed using the **mfspr**/**mtspr** instructions. Please refer to the *e200z6 Power Architecture*™ *Core Reference Manual* for more details on the process ID register.

One condition causes an ownership trace message: When new information is updated in the OTR register or process ID register by the e200z6 processor, the data is latched within Nexus, and is messaged out via the auxiliary port, allowing development tools to trace ownership flow.

Ownership trace information is messaged out in the following format:

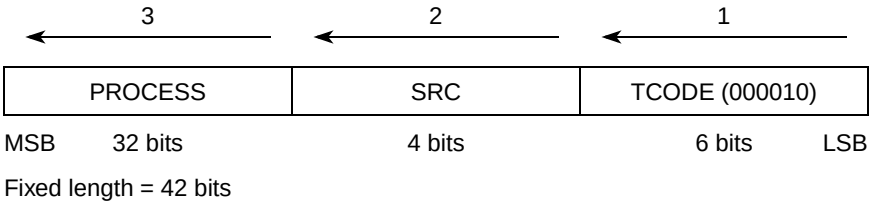


Figure 36-26. Ownership Trace Message Format

36.6.10.2.3 OTM Error Messages

An error message occurs when a new message cannot be queued due to the message queue being full. The FIFO discards incoming messages until it has completely emptied the queue. Once emptied, an error message is queued. The error encoding indicates which types of messages attempted to be queued while the FIFO was being emptied.

If only an OTM message attempts to enter the queue while it is being emptied, the error message incorporates the OTM only error encoding (00000). If both OTM and either BTM or DTM messages attempt to enter the queue, the error message incorporates the OTM and (program or data) trace error encoding (00111). If a watchpoint also attempts to be queued while the FIFO is being emptied, then the error message incorporates error encoding (01000).

NOTE

The OVC bits within the DC1 register can be set to delay the CPU in order to alleviate (but not eliminate) potential overrun situations.

Error information is messaged out in the following format (see Table 36-15)

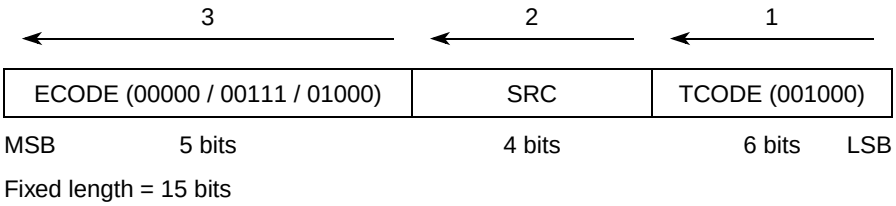


Figure 36-27. Error Message Format

36.6.10.2.4 OTM Flow

Ownership trace messages are generated when the operating system writes to the e200z6 process ID register or the memory mapped ownership trace register.

The following flow describes the OTM process:

1. The process ID register is a system control register. It is internal to the e200z6 processor and can be accessed by using PPC instructions **mtspr** and **mfspr**. The contents of this register are replicated on the pins of the processor and connected to Nexus.
2. OTR/process ID register reads do not cause ownership trace messages to be transmitted by the Nexus3+ module.
3. If the periodic OTM message counter expires (after 255 queued messages without an OTM), an OTM is sent using the latched data from the previous OTM or process ID register write.

36.6.10.3 Program Trace

This section details the program trace mechanism supported by Nexus3+ for the e200z6 processor. Program trace is implemented via branch trace messaging (BTM) as per the Class 3 IEEE-ISTO 5001-2003 standard definition. Branch trace messaging for e200z6 processors is accomplished by snooping the e200z6 virtual address bus (between the CPU and MMU), attribute signals, and CPU status.

36.6.10.3.1 Branch Trace Messaging (BTM)

Traditional branch trace messaging facilitates program trace by providing the following types of information:

- Messaging for taken direct branches includes how many sequential instructions were executed since the last taken branch or exception. Direct (or indirect) branches not taken are counted as sequential instructions.
- Messaging for taken indirect branches and exceptions includes how many sequential instructions were executed since the last taken branch or exception and the unique portion of the branch target address or exception vector address.

Branch history messaging facilitates program trace by providing the following information:

- Messaging for taken indirect branches and exceptions includes how many sequential instructions were executed since the last predicate instruction, taken indirect branch, or exception, the unique portion of the branch target address or exception vector address, as well as a branch/predicate instruction history field. Each bit in the history field represents a direct branch or predicated instruction where a value of one (1) indicates taken, and a value of zero (0) indicates not taken. Certain instructions (**evsel**) generate a pair of predicate bits that are both reported as consecutive bits in the history field.

e200z6 Indirect Branch Message Instructions (Power Architecture Book E)

Table 36-31 shows the types of instructions and events that cause indirect branch messages or branch history messages to be encoded.

Table 36-31. Indirect Branch Message Sources

Source of Indirect Branch Message	Instructions
Taken branch relative to a register value	**bcctr**, **bcctrl**, **bclr**, **bclrl, se_bctr, se_bctrl, se_blr, se_blrl**
System Call / Trap exceptions taken	**sc**, **tw**, **twi, se_sc**
Return from interrupts / exceptions	**rfi**, **rfci**, **rfdi, se_rfi, se_rfci, se_rfdi**

e200z6 Direct Branch Message Instructions (Power Architecture Book E)

Table 36-32 shows the types of instructions that cause direct branch messages or toggle a bit in the instruction history buffer to be messaged out in a resource full message or branch history message.

Table 36-32. Direct Branch Message Sources

Source of Direct Branch Message	Instructions
Taken direct branch instructions	**b**, **ba**, **bl**, **bla**, **bc**, **bca**, **bcl**, **bcla, se_b. se_bc, se_bl, e_b, e_bc, e_bl, e_bcl,**
Instruction Synchronize	isync, se_isync

BTM Using Branch History Messages

Traditional BTM messaging can accurately track the number of sequential instructions between branches, but cannot accurately indicate which instructions were conditionally executed, and which were not.

Branch history messaging solves this problem by providing a predicated instruction history field in each indirect branch message. Each bit in the history represents a predicated instruction or direct branch. A value of one (1) indicates the conditional instruction was executed or the direct branch was taken. A value of zero (0) indicates the conditional instruction was not executed or the direct branch was not taken. Certain instructions (**evsel**) generate a pair of predicate bits that are both reported as consecutive bits in the history field.

Branch history messages solve predicated instruction tracking and save bandwidth since only indirect branches cause messages to be queued.

BTM Using Traditional Program Trace Messages

Based on the PTM bit in the DC register (DC[PTM]), program tracing can utilize either branch history messages (DC[PTM] = 1) or traditional direct/indirect branch messages (DC[PTM] = 0).

Branch history saves bandwidth and keeps consistency between methods of program trace, yet may lose temporal order between BTM messages and other types of messages. Since direct branches are not messaged, but are instead included in the history field of the indirect branch history message, other types of messages may enter the FIFO between branch history messages. The development tool cannot determine the ordering of “events” that occurred with respect to direct branches simply by the order in which messages are sent out.

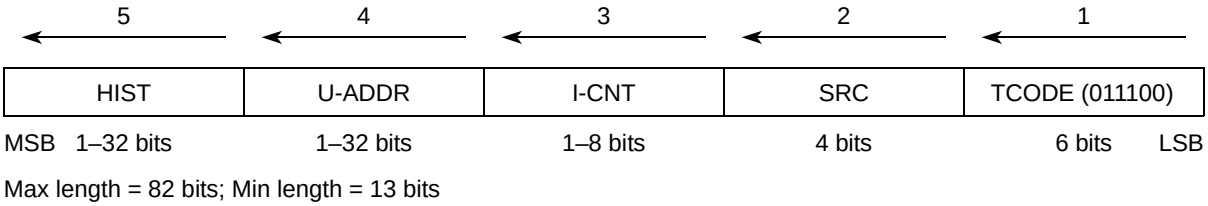
Traditional BTM messages maintain their temporal ordering because each event that can cause a message to be queued enters the FIFO in the order it occurred and is messaged out in that same order.

36.6.10.3.2 BTM Message Formats

The e200z6 Nexus3 module supports three types of traditional BTM messages—direct, indirect, and synchronization messages. It supports two types of branch history BTM messages—indirect branch history, and indirect branch history with synchronization messages. Debug status messages and error messages are also supported.

36.6.10.3.3 Indirect Branch Messages (History)

Indirect branches include all taken branches whose destination is determined at run time, interrupts and exceptions. If DC[PTM] is set, indirect branch information is messaged out in the following format:

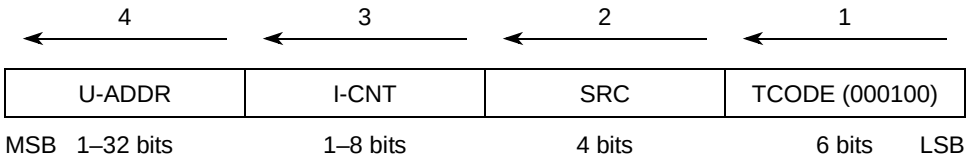


Max length = 82 bits; Min length = 13 bits

Figure 36-28. Indirect Branch Message (History) Format

Indirect Branch Messages (Traditional)

If DC[PTM] is cleared, indirect branch information is messaged out in the following format:

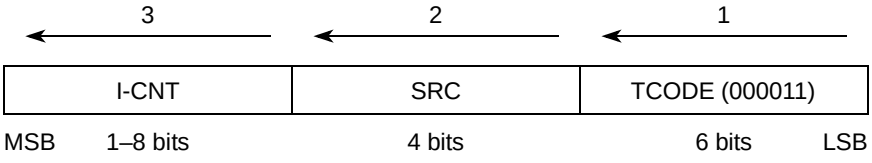


Max length = 50 bits; Min length = 12 bits

Figure 36-29. Indirect Branch Message Format

Direct Branch Messages (Traditional)

Direct branches (conditional or unconditional) are all taken branches whose destination is fixed in the instruction opcode. Direct branch information is messaged out in the following format:



Max length = 18 bits; Min length = 11 bits

Figure 36-30. Direct Branch Message Format

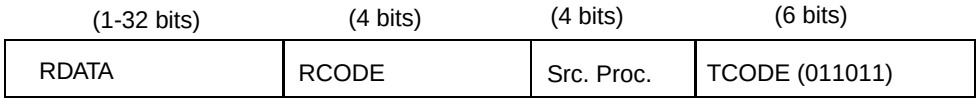
NOTE

When DC[PTM] is set, direct branch messages are not transmitted. Instead, each direct branch or predicated instruction toggles a bit in the history buffer.

Resource Full Messages

The resource full message is used in conjunction with the branch history messages. The resource full message is generated when the internal branch/predicate history buffer is full, or if the BTM Instruction sequence counter (I-CNT) overflows. If synchronization is needed at the time this message is generated, the synchronization is delayed until the next branch trace message that is not a resource full message.

The current value of the history buffer is transmitted as part of the resource full message. This information can be concatenated by the tool with the branch/predicate history information from subsequent messages to obtain the complete branch history for a message. The internal history value is reset by this message, and the I-CNT value is reset as a result of a bit being added to the history buffer.



Max length = 46 bits; Min length = 15 bits

Figure 36-31. Resource Full Message Format

Table 36-33 shows the RCODE encodings and RDATA information used for Resource Full messages

Table 36-33. RCODE Encoding

RCODE	Description	RDATA field
0000	Program Trace Instruction counter reached 255 and was reset.	0xFF
0001	Program Trace, Branch / Predicate Instruction History full.	Branch HIstory. This type of packet is terminated by a stop bit set to 1 after the last history bit.

Program Correlation Messages

Program correlation messages are used to correlate events to the program flow that may not be associated with the instruction stream. The following events result in a PCM when program trace is enabled:

- When the CPU enters debug mode, a PCM is generated. The instruction count and history information provided by the PCM can be used to determine the last sequence of instructions executed prior to debug mode entry.
- When the CPU enters a low power mode in which instructions are no longer executed, a PCM is generated. The instruction count and history information provided by the PCM can be used to determine the last sequence of instructions executed prior to low power mode entry.
- Whenever program trace is disabled by any means, a PCM is generated. The instruction count and history information provided by the PCM can be used to determine the last sequence of instructions executed prior to disabling program trace. A second PCM is generated on this event if there has

been an execution mode switch into or out of a sequence of VLE instructions. This VLE state information allows the development tool to interpret any preceding instruction count or history information in the proper context.

- Whenever the CPU crosses a page boundary that results in an execution mode switch into or out of a sequence of VLE instructions, a PCM is generated. The PCM effectively breaks up any running instruction count and history information between the two modes of operation so that the instruction count and history information can be processed by the development tool in the proper context.
- When using program trace in history mode, when a direct branch results in an execution mode switch into or out of a sequence of VLE instructions, a PCM is generated. The PCM effectively breaks up any running history information between the two modes of operation so that the history information can be processed by the development tool in the proper context.

Program correlation is messaged out in the following format:

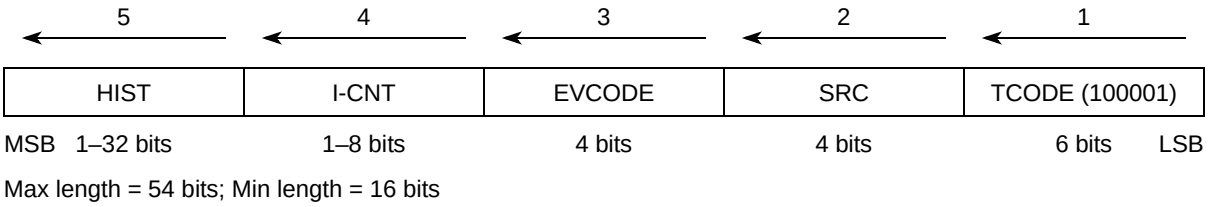


Figure 36-32. Program Correlation Message Format

BTM Overflow Error Messages

An error message occurs when a new message cannot be queued due to the message queue being full. The FIFO discards incoming messages until it has completely emptied the queue. Once emptied, an error message is queued. The error encoding indicates which types of messages attempted to be queued while the FIFO was being emptied.

If only a program trace message attempts to enter the queue while it is being emptied, the error message incorporates the program trace only error encoding (00001). If both OTM and program trace messages attempt to enter the queue, the error message incorporates the OTM and program trace error encoding (00111). If a watchpoint also attempts to be queued while the FIFO is being emptied, then the error message incorporates error encoding (01000).

NOTE

The OVC bits within the DC1 register can be set to delay the CPU in order to alleviate (but not eliminate) potential overrun situations.

Error information is messaged out in the following format

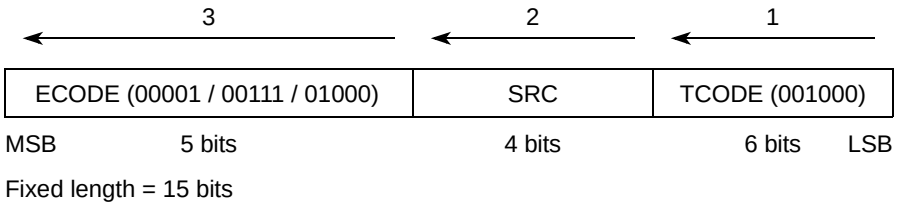


Figure 36-33. Error Message Format

Program Trace Synchronization Messages

A program trace direct/indirect branch with sync message is messaged via the auxiliary port (provided program trace is enabled) for the following conditions (see Table 36-34):

- Initial program trace message upon the first direct/indirect branch after exit from system reset or whenever program trace is enabled
- Upon direct/indirect branch after returning from a CPU low power state
- Upon direct/indirect branch after returning from debug mode
- Upon direct/indirect branch after occurrence of queue overrun (can be caused by any trace message), provided program trace is enabled
- Upon direct/indirect branch after the periodic program trace counter has expired indicating 255 *without-sync* program trace messages have occurred since the last *with-sync* message occurred
- Upon direct/indirect branch after assertion of the event in ($\overline{\text{EVTI}}$) pin if the EIC bits within the DC1 register have enabled this feature
- Upon direct/indirect branch after the sequential instruction counter has expired indicating 255 instructions have occurred between branches
- Upon direct/indirect branch after a BTM message was lost due to an attempted access to a secure memory location.
- Upon direct/indirect branch after a BTM message was lost due to a collision entering the FIFO between the BTM message and either a watchpoint message or an ownership trace message
- Upon the first direct/indirect branch message after an execution mode switch

If the Nexus3+ module is enabled at reset, a $\overline{\text{EVTI}}$ assertion initiates a program trace direct/indirect branch with sync message (if program trace is enabled) upon the first direct/indirect branch. The format for program trace direct/indirect branch with sync messages is as follows:

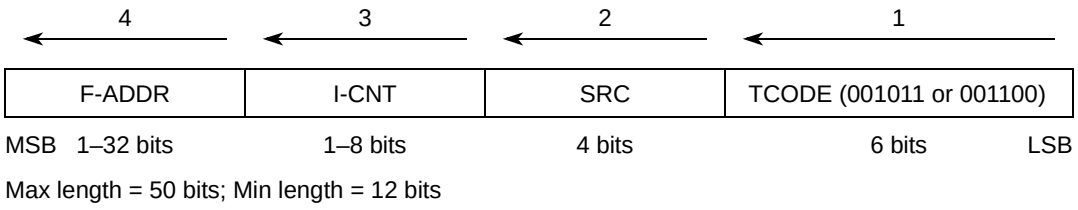


Figure 36-34. Direct/Indirect Branch with Sync Message Format

The formats for program trace direct/indirect branch with sync. messages and indirect branch history with sync. messages are as follows

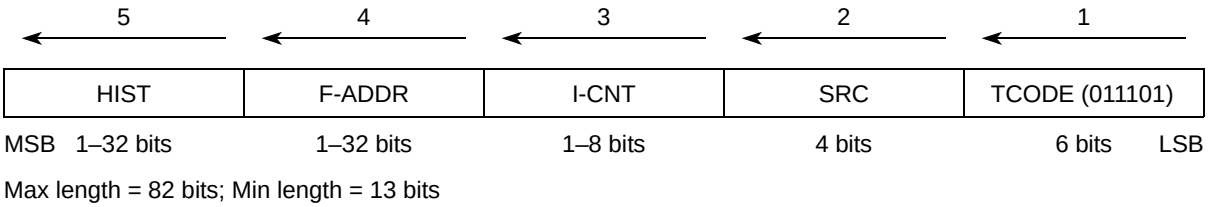


Figure 36-35. Indirect Branch History with Sync. Message Format

Exception conditions that result in program trace synchronization are summarized in Table 36-34.

Table 36-34. Program Trace Exception Summary

Exception Condition	Exception Handling
System Reset Negation	At the negation of JTAG reset (JCOMP), queue pointers, counters, state machines, and registers within the Nexus3+ module are reset. Upon the first branch out of system reset (if program trace is enabled), the first program trace message is a direct/indirect branch with sync. message.
Program Trace Enabled	The first program trace message (after program trace has been enabled) is a synchronization message.
Exit from Low Power/Debug	Upon exit from a low power mode or debug mode, the next direct/indirect branch is converted to a direct/indirect branch with sync. message.
Queue Overrun	An error message occurs when a new message cannot be queued due to the message queue being full. The FIFO discards messages until it has completely emptied the queue. Once emptied, an error message is queued. The error encoding indicates which types of messages attempted to be queued while the FIFO was being emptied. The next BTM message in the queue is a direct/indirect branch with sync. message.
Periodic Program Trace Sync.	A forced synchronization occurs periodically after 255 program trace messages have been queued. A direct/indirect branch with sync. message is queued. The periodic program trace message counter then resets.
Event In	If the Nexus module is enabled, an $\overline{\text{EVTI}}$ assertion initiates a direct/indirect branch with sync. message upon the next direct/indirect branch (if program trace is enabled and the EIC bits of the DC1 register have enabled this feature).
Sequential Instruction Count Overflow	When the sequential instruction counter reaches its maximum count (as many as 255 sequential instructions may be executed), a forced synchronization occurs. The sequential counter then resets. A program trace direct/indirect branch with sync.message is queued upon execution of the next branch.
Attempted Access to Secure Memory	For devices that implement security, any attempted branch to secure memory locations temporarily disables program trace, and causes the corresponding BTM to be lost. The following direct/indirect branch queues a direct/indirect branch with sync. message. The count value within this message is inaccurate since the re-enable of program trace is not necessarily aligned on an instruction boundary.
Collision Priority	All messages have the following priority: WPM → OTM → BTM → DTM. A BTM message that attempts to enter the queue at the same time as a watchpoint message or ownership trace message is lost. An error message is sent indicating the BTM was lost. The following direct/indirect branch queues a direct/indirect branch with sync. message. The count value within this message reflects the number of sequential instructions executed after the last successful BTM Message was generated. This count includes the branch that did not generate a message due to the collision.
Execution Mode Switch	Whenever the CPU switches execution mode into or out of a sequence of VLE instructions, the next branch trace message is a Direct/Indirect Branch w/ Sync Message.

36.6.10.3.4 BTM Operation

Enabling Program Trace

Both types of branch trace messaging are enabled using one of the following methods:

- Setting the TM field of the DC1 register to enable program trace (DC1[TM])
- Using the PTS field of the WT register to enable program trace on watchpoint hits (e200z6 watch points are configured within the CPU)

Relative Addressing

The relative address feature is compliant with the IEEE-ISTO 5001-2003 standard recommendations, and is designed to reduce the number of bits transmitted for addresses of indirect branch messages.

The address transmitted is relative to the target address of the instruction that triggered the previous indirect branch (or sync) message. It is generated by XOR-ing the new address with the previous address, and then using only the results up to the most significant 1 in the result. To recreate this address, an XOR of the (most-significant 0-padded) message address with the previously decoded address gives the current address.

Previous address (A1) = 0x0003_FC01, New address (A2) = 0x0003_F365

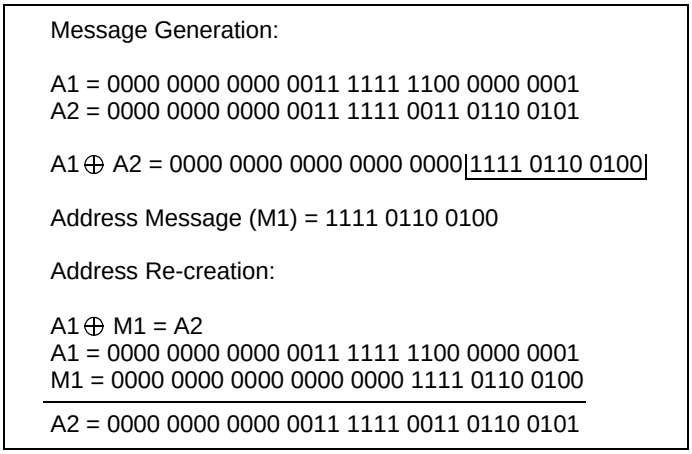


Figure 36-36. Relative Address Generation and Re-creation

Execution Mode Indication

In order for a development tool to properly interpret instruction count and history information, it must be aware of the execution mode context of that information. VLE instructions are interpreted differently from non-VLE instructions.

Program trace messages provide the execution mode status in the least significant bit of the reconstructed address field. A value of ‘0’ indicates that preceding instruction count and history information should be interpreted in a non-VLE context. A value of ‘1’ indicates that the preceding instruction count and history information should be interpreted in a VLE context. Note that when a branch results in an execution mode

switch, the program trace message resulting from that branch indicates the previous execution state. The new state is not signaled until the next program trace message.

In some cases, a Program Correlation Message is generated to indicate execution mode status Refer to Program Correlation Messages on page 36-88 for more information on these cases.

Branch/Predicate Instruction History (HIST)

If DC[PTM] is set, BTM messaging uses the branch history format. The branch history (HIST) packet in these messages provides a history of direct branch execution used for reconstructing the program flow. This packet is implemented as a left-shifting shift register. The register is always pre-loaded with a value of one (1). This bit acts as a stop bit so that the development tools can determine which bit is the end of the history information. The pre-loaded bit itself is not part of the history, but is transmitted with the packet.

A value of one (1) is shifted into the history buffer on a taken branch (condition or unconditional) and on any instruction whose predicate condition executed as true. A value of zero (0) is shifted into the history buffer on any instruction whose predicate condition executed as false as well as on branches not taken. This includes indirect as well as direct branches not taken. For the **evsel** instruction, two bits are shifted in, corresponding to the low element (shifted in first) and the high element (shifted in second) conditions.

Sequential Instruction Count (I-CNT)

The I-CNT packet is present in all BTM messages. For traditional branch messages, I-CNT represents the number of sequential instructions, or non-taken branches in between direct/indirect branch messages.

For branch history messages, I-CNT represents the number of instructions executed since the last taken/non-taken direct branch, last taken indirect branch or exception. Not taken indirect branches are considered sequential instructions and cause the instruction count to increment. I-CNT also represents the number of instructions executed since the last predicate instruction.

The sequential instruction counter overflows when its value reaches 255. The next BTM message is converted to a synchronization type message.

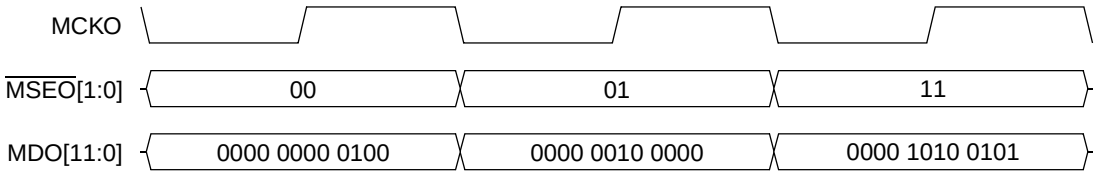
Program Trace Queueing

Nexus3+ implements a message queue. Messages that enter the queue are transmitted via the auxiliary pins in the order in which they are queued.

NOTE

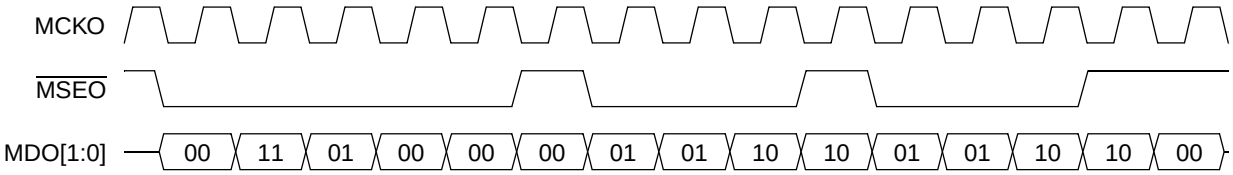
If multiple trace messages need to be queued at the same time, Watchpoint Messages have the highest priority (WPM –> OTM –> BTM –> DTM).

36.6.10.3.5 Program Trace Timing Diagrams



TCODE = 4
Source Processor = 0b0000
Number of Sequence Instructions = 128
Relative Address = 0xA5

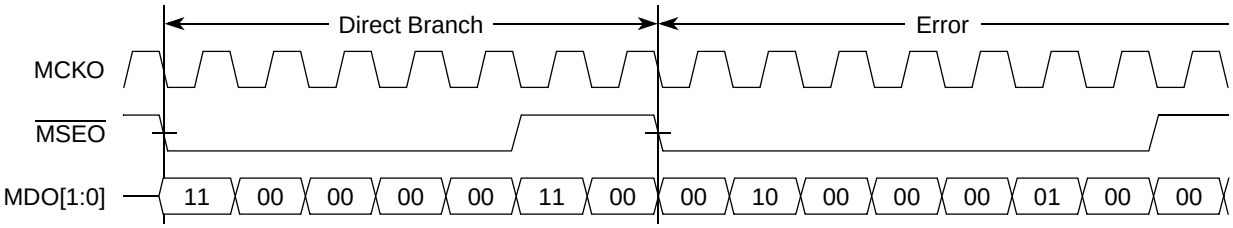
Figure 36-37. Program Trace (MDO = 12)—Indirect Branch Message (Traditional)



TCODE = 28
Source Processor = 0b0000
Number of Sequential Instructions = 0
Relative Address = 0xA5
Branch History = 0b1010_0101 (with Stop)

Note: This is representative only. The PXN20 supports only Full-Port Mode with 12 $\overline{\text{MDO}}$ pins.

Figure 36-38. Program Trace (MDO = 2)—Indirect Branch Message (History)

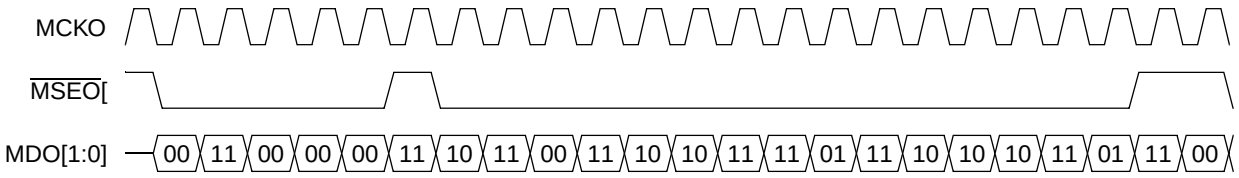


DBM:
TCODE = 3
Source Processor = 0b0000
Number of Sequential Instructions = 3

Error:
TCODE = 8
Source Processor = 0b0000
Error Code = 1 (Queue Overrun – BTM Only)

Note: This is representative only. The PXN20 x supports only Full-Port Mode with 12 $\overline{\text{MDO}}$ pins.

Figure 36-39. Program Trace—Direct Branch (Traditional) and Error Messages



TCODE = 12
Source Processor = 0b0000
Number of Sequential Instructions = 3
Full Target Address = 0xDEAD_FACE

Note: This is representative only. The PXN20 supports only Full-Port Mode with 12 $\overline{\text{MDO}}$ pins.

Figure 36-40. Program Trace—Indirect Branch with Sync. Message

36.6.10.4 Data Trace

This section deals with the data trace mechanism supported by the Nexus3+ module. Data trace is implemented via data write messaging (DWM) and data read messaging (DRM), as per the IEEE-ISTO 5001-2003 standard.

36.6.10.4.1 Data Trace Messaging (DTM)

Data trace messaging for e200z6 is accomplished by snooping the e200z6 virtual data bus (between the CPU and MMU), and storing the information for qualifying accesses (based on enabled features and matching target addresses). The Nexus3+ module traces all data access that meet the selected range and attributes.

NOTE

Data trace is only performed on the e200z6 virtual data bus. This allows for data visibility for the incorporated data cache. Only e200z6 CPU initiated accesses are traced.

Data trace messaging can be enabled in one of two ways:

- Setting the TM field of the DC1 register to enable data trace (DC1[TM]).
- Using WT[DTS] to enable data trace on watchpoint hits (e200z6 watch points are configured within the Nexus1 module)

36.6.10.4.2 DTM Message Formats

The Nexus3 module supports five types of DTM messages: data write, data read, data write synchronization, data read synchronization and error messages.

Data Write Messages

The data write message contains the data write value and the address of the write access, relative to the previous data trace message. Data write message information is messaged out in the following format:

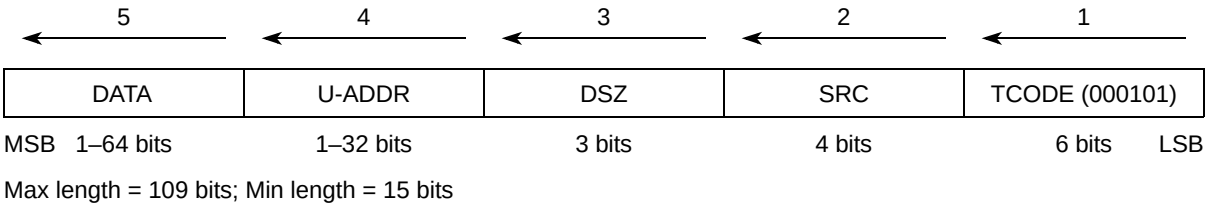


Figure 36-41. Data Write Message Format

Data Read Messages

The data read message contains the data read value and the address of the read access, relative to the previous data trace message. Data read message information is messaged out in the following format:

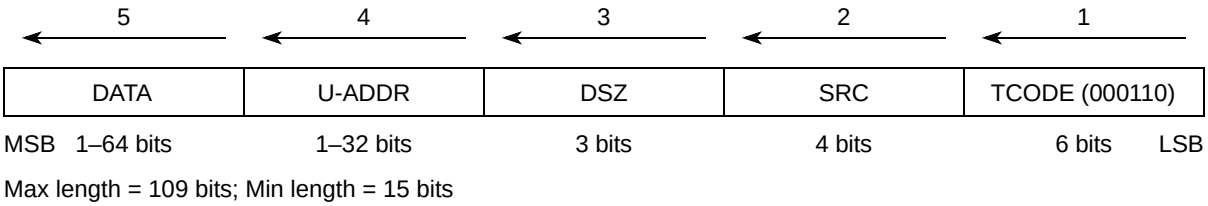


Figure 36-42. Data Read Message Format

NOTE

The e200z6 Z5XX based CPUs are capable of generating two (2) reads or writes per clock cycle in cases where multiple registers are accessed with a single instruction (lmw/stmw). These have a double word pair size encoding (**DSZ** = 0b000). In these cases, the Nexus3 module sends one (1) Data Trace Message with the two 32-bit data values as one combined 64-bit value for each message.

For the e200z6 based CPU, the double-word encoding (data size = 0b000) indicates a double-word access and is sent out as a single data trace message with a single 64-bit data value.

The debug/development tool needs to distinguish the two cases based on the family of Zen processor.

DTM Overflow Error Messages

An error message occurs when a new message cannot be queued due to the message queue being full. The FIFO discards incoming messages until it has completely emptied the queue. Once emptied, an error message is queued. The error encoding indicates which types of messages attempted to be queued while the FIFO was being emptied.

If only a data trace message attempts to enter the queue while it is being emptied, the error message incorporates the data trace only error encoding (00010). If both OTM and data trace messages attempt to enter the queue, the error message incorporates the OTM and data trace error encoding (00111). If a watchpoint also attempts to be queued while the FIFO is being emptied, then the error message incorporates error encoding (01000).

NOTE

The OVC bits within the DC1 register can be set to delay the CPU in order to alleviate (but not eliminate) potential overrun situations.

Error information is messaged out in the following format:

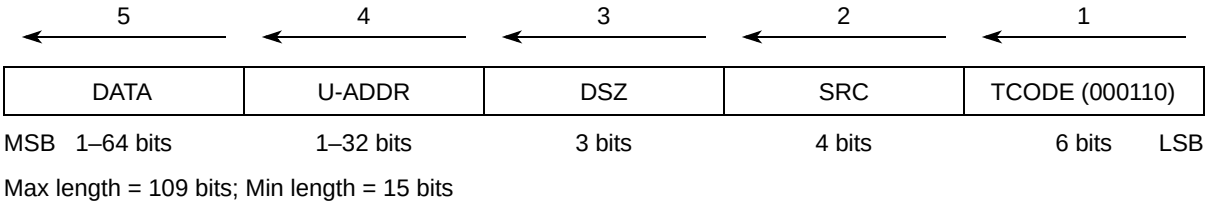


Figure 36-43. Error Message Format

Data Trace Synchronization Messages

A data trace write/read with sync. message is messaged via the auxiliary port (provided data trace is enabled) for the following conditions (see Table 36-35):

- Initial data trace message after exit from system reset or whenever data trace is enabled
- Upon returning from a CPU Low Power state.
- Upon exiting debug mode
- After occurrence of queue overrun (can be caused by any trace message), provided data trace is enabled
- After the periodic data trace counter has expired indicating 255 *without-sync* data trace messages have occurred since the last *with-sync* message occurred
- Upon assertion of the event in ($\overline{\text{EVTI}}$) pin, the first data trace message is a synchronization message if the EIC bits of the DC1 register have enabled this feature
- Upon data trace write/read after the previous DTM message was lost due to an attempted access to a secure memory location
- Upon data trace write/read after the previous DTM message was lost due to a collision entering the FIFO between the DTM message and any of the following: watchpoint message, ownership trace message, or branch trace message

Data trace synchronization messages provide the full address (without leading zeros) and ensure that development tools fully synchronize with data trace regularly. Synchronization messages provide a reference address for subsequent data messages, in which only the unique portion of the data trace address is transmitted. The format for data trace write/read with sync. messages is as follows:

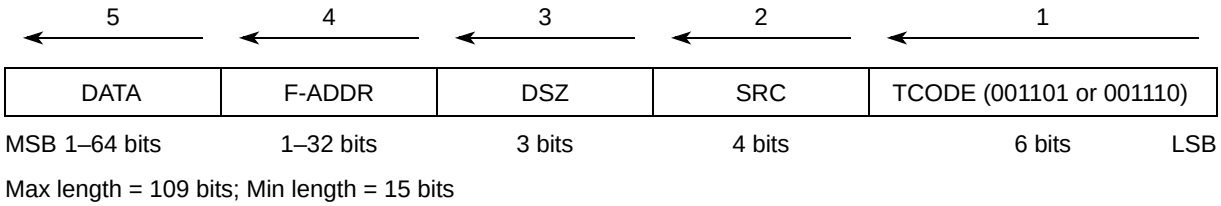


Figure 36-44. Data Write/Read with Sync. Message Format

Exception conditions that result in data trace synchronization are summarized in Table 36-35.

Table 36-35. Data Trace Exception Summary

Exception Condition	Exception Handling
System Reset Negation	At the negation of JTAG reset (JCOMP), queue pointers, counters, state machines, and registers within the Nexus3+ module are reset. If data trace is enabled, the first data trace message is a data write/read with sync. message.
Data Trace Enabled	The first data trace message (after data trace has been enabled) is a synchronization message.
Exit from Low Power/Debug	Upon exit from a low power mode or debug mode, the next data trace message is converted to a data write/read with sync. message.
Queue Overrun	An error message occurs when a new message cannot be queued due to the message queue being full. The FIFO discards messages until it has completely emptied the queue. Once emptied, an error message is queued. The error encoding indicates which types of messages attempted to be queued while the FIFO was being emptied. The next DTM message in the queue is a data write/read with sync. message.
Periodic Data Trace Sync.	A forced synchronization occurs periodically after 255 data trace messages have been queued. A data write/read with sync. message is queued. The periodic data trace message counter then resets.
Event In	If the Nexus module is enabled, a $\overline{\text{EVTI}}$ assertion initiates a data trace write/read with sync. message upon the next data write/read (if data trace is enabled and the EIC bits of the DC1 register have enabled this feature).
Attempted Access to Secure Memory	For devices that implement security, any attempted read or write to secure memory locations temporarily disables data trace and causes the corresponding DTM to be lost. A subsequent read/write queues a data trace read/write with sync. message.
Collision Priority	All messages have the following priority: WPM –> OTM –> BTM –> DTM. A DTM message that attempts to enter the queue at the same time as a watchpoint message or ownership trace message or branch trace message is lost. A subsequent read/write queues a data trace read/write with sync. message.

36.6.10.4.3 DTM Operation

DTM Queueing

Nexus3+ implements a message queue for DTM messages. Messages that enter the queue are transmitted via the auxiliary pins in the order in which they are queued.

NOTE

If multiple trace messages need to be queued at the same time, watchpoint messages have the highest priority (WPM → OTM → BTM → DTM).

Relative Addressing

The relative address feature is compliant with the IEEE-ISTO 5001-2003 standard recommendations, and is designed to reduce the number of bits transmitted for addresses of data trace messages. Refer to Relative Addressing for details.

Data Trace Windowing

Data write/read messages are enabled via the RWT1(2) field in the data trace control register (DTC) for each DTM channel. Data trace windowing is achieved via the address range defined by the DTEA and DTSA registers and by the RC1(2) field in the DTC. All e200z6 initiated read/write accesses that fall inside or outside these address ranges, as programmed, are candidates to be traced.

Data Access/Instruction Access Data Tracing

The Nexus3 module is capable of tracing both instruction access data or data access data. Each trace window can be configured for either type of data trace by setting the DI1(2) field within the data trace control register for each DTM channel.

e200z6 Bus Cycle Special Cases

Table 36-36. e200z6 Bus Cycle Cases

Special Case	Action
e200z6 bus cycle aborted	Cycle ignored
e200z6 bus cycle with data error ($\overline{\text{TEA}}$)	Data Trace Message discarded
e200z6 bus cycle completed without error	Cycle captured & transmitted
e200z6 bus cycle initiated by Nexus3+	Cycle ignored
e200z6 bus cycle is an instruction fetch	Cycle ignored
e200z6 bus cycle accesses misaligned data (across 64-bit boundary)—both 1st & 2nd transactions within data trace range	1st & 2nd cycle captured, and 2 DTMs transmitted (see Note)
e200z6 bus cycle accesses misaligned data (across 64-bit boundary)—1st transaction within data trace range; 2nd transaction out of data trace range	1st cycle captured and transmitted; 2nd cycle ignored
e200z6 bus cycle accesses misaligned data (across 64-bit boundary)—1st transaction out of data trace range; 2nd transaction within data trace range	1st cycle ignored; 2nd cycle capture and transmitted

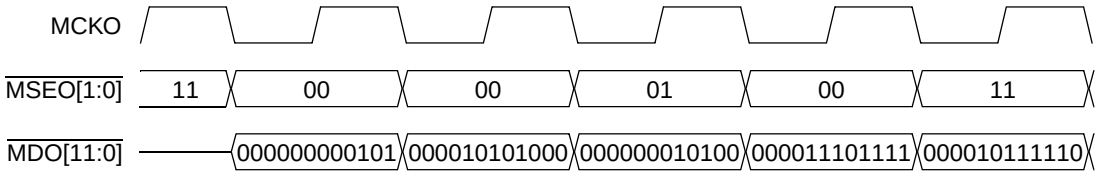
NOTE

For misaligned accesses (crossing 64-bit boundary), the access is broken into two accesses. If both accesses are within the data trace range, two DTMs are sent: one with a size encoding indicating the size of the original access (that is, word), and one with a size encoding for the portion that crossed the boundary (that is, 3-byte).

NOTE

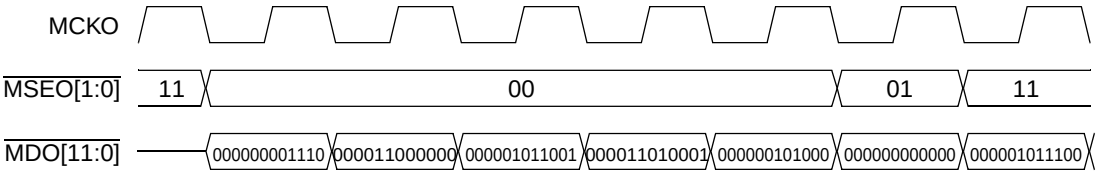
An STM to the cache's store buffer within the data trace range initiates a DTM message. If the corresponding memory access causes an error, a checkstop condition occurs. Use the checkstop condition in the debug/development tool to invalidate the previous DTM.

36.6.10.4.4 Data Trace Timing Diagrams (12 MDO Configuration)



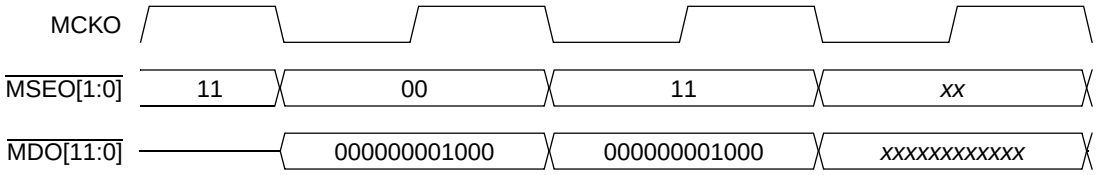
TCODE = 5
Source Processor = 0b0000
Data Size = 010 (Half-Word)
Relative Address = 0xA5
Write Data = 0xBEEF

Figure 36-45. Data Trace—Data Write Message



TCODE = 14
Source Processor = 0b0000
Data Size = 000 (Byte)
Full Access Address = 0x0146_8ACE
Write Data = 0x5C

Figure 36-46. Data Trace—Data Read with Sync Message



TCODE = 8
Source Processor = 0b0000
Error Code = 2 (Queue Overrun – DTM Only)

Figure 36-47. Error Message (Data Trace only encoded)

36.6.10.5 Watchpoint Support

This section details the watchpoint features of the Nexus3+ module.

36.6.10.5.1 Overview

The Nexus3+ module provides watchpoint messaging via the auxiliary pins, as defined by the IEEE-ISTO 5001-2003 standard.

Nexus3+ is not compliant with Class4 breakpoint/watchpoint requirements defined in the standard. The breakpoint/watchpoint control register is not implemented.

36.6.10.5.2 Watchpoint Messaging

Enabling watchpoint messaging is done by setting the watchpoint enable bit in the DC1 register. Setting the individual watchpoint sources is supported through the e200z6 Nexus1 module. The e200z6 Nexus1 module is capable of setting multiple address and/or data watch points. Please refer to the *e200z6 Core Reference Manual* for more information on watchpoint initialization.

When these watch points occur, a watchpoint event signal from the Nexus1 module causes a message to be sent to the queue to be messaged out. This message includes the watchpoint number indicating which watchpoint caused the message.

The occurrence of any of the e200z6 defined watch points can be programmed to assert the event out $\overline{\text{EVTO}}$ pin for one (1) period of the output clock (MCKO).

Watchpoint information is messaged out in the following format

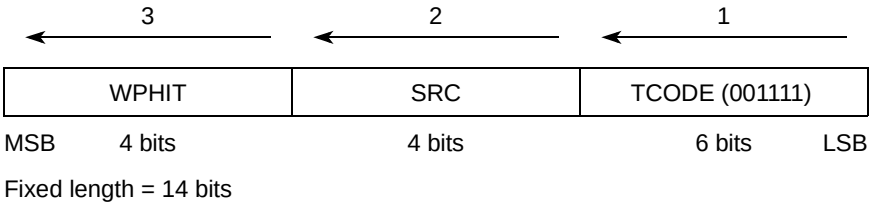


Figure 36-48. Watchpoint Message Format.

Table 36-37. Watchpoint Source Encoding

Watchpoint Source (8 bits)	Watchpoint Description
0b0000_0001	e200z6 Watchpoint #0 (IAC1 from Nexus1)
0b0000_0010	e200z6 Watchpoint #1 (IAC2 from Nexus1)
0b0000_0100	e200z6 Watchpoint #2 (IAC3 from Nexus1)
0b0000_1000	e200z6 Watchpoint #3 (IAC4 from Nexus1)
0b0001_0000	e200z6 Watchpoint #4 (DAC1 from Nexus1)
0b0010_0000	e200z6 Watchpoint #5 (DAC2 from Nexus1)
0b0100_0000	e200z6 Watchpoint #6 (DCNT1 from Nexus1)
0b1000_0000	e200z6 Watchpoint #7 (DCNT2 from Nexus1)

36.6.10.5.3 Watchpoint Error Message

An error message occurs when a new message cannot be queued due to the message queue being full. The FIFO discards messages until it has completely emptied the queue. Once emptied, an error message is queued. The error encoding indicates the type of messages queued to the FIFO while it was emptying.

If only a watchpoint message attempts to enter the queue while it is being emptied, the error message incorporates the watchpoint only error encoding (00110). If an OTM and/or program trace and/or data trace message also attempts to enter the queue while it is being emptied, the error message incorporates error encoding (01000).

NOTE

Set the OVC bits within the DC1 register to delay the CPU to alleviate (but not eliminate) potential overrun situations.

Error information is messaged out in the following format (see Table 36-15)

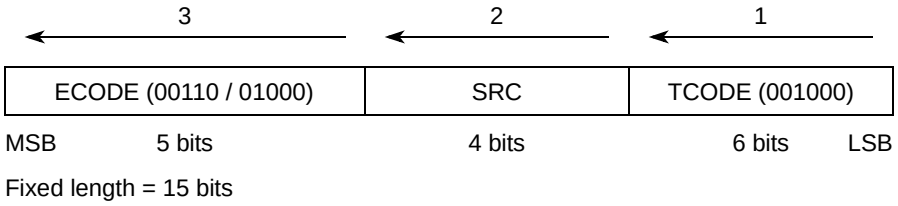
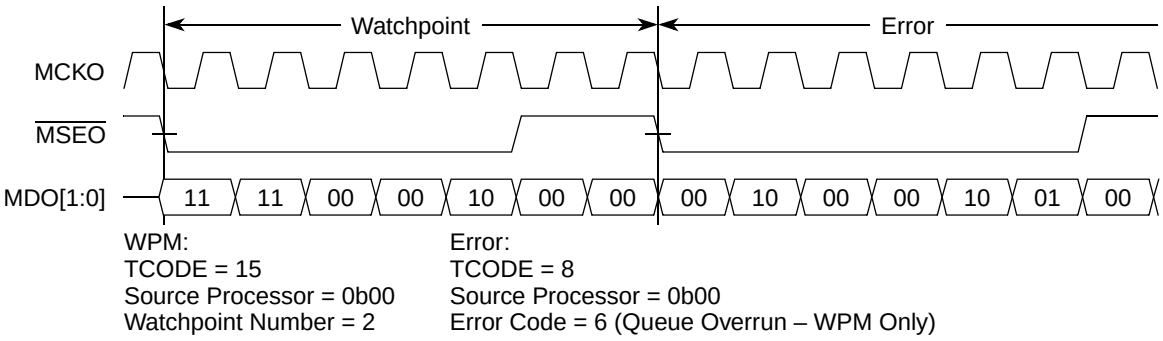


Figure 36-49. Error Message Format

36.6.10.5.4 Watchpoint Timing Diagram (2 MDO/1 MSEO Configuration)



Note: This is representative only. The PXN20 supports only Full-Port Mode with 12 $\overline{\text{MDO}}$ pins.

Figure 36-50. Watchpoint Message and Watchpoint Error Message

36.6.10.6 Nexus3+ Read/Write Access to Memory-Mapped Resources

The read/write access feature allows access to memory-mapped resources via the JTAG/OnCE port. The read/write mechanism supports single as well as block reads and writes to e200z6 system bus resources.

The Nexus3+ module is capable of accessing resources on the e200z6 system bus, with multiple configurable priority levels. Memory-mapped registers and other non-cached memory can be accessed via the standard memory map settings.

All accesses are setup and initiated by the read/write access control/status register (RWCS), as well as the read/write access address (RWA) and read/write access data registers (RWD).

Using the read/write access registers (RWCS/RWA/RWD), memory-mapped e200z6 system bus resources can be accessed through Nexus3+. The following subsections describe the steps required to access memory-mapped resources.

NOTE

Read/write access can only access memory mapped resources when system reset is de-asserted.

Misaligned accesses are NOT supported in the e200z6 Nexus3 module.

36.6.10.6.1 Single Write Access

1. Initialize the read/write access address register (RWA) through the access method outlined in Section 36.6.9, Nexus3+ Register Access via JTAG / OnCE using the Nexus register index of 0x9 (see Table 36-19). Configure as follows:
   - Write Address → 0x*nnnn_nnnn* (write address)
2. Initialize the read/write access control/status register (RWCS) through the access method outlined in Section 36.6.9, Nexus3+ Register Access via JTAG / OnCE, using the Nexus Register Index of 0x7(see Table 36-19). Configure the bits as follows:
   - Access Control RWCS[AC] → 0b1 (to indicate start access)
   - Map Select RWCS[MAP] → 0b000 (primary memory map)
   - Access Priority RWCS[PR] → 0b00 (lowest priority)
   - Read/Write RWCS[RW] → 0b1 (write access)
   - Word Size RWCS[SZ] → 0b0*xx* (32-bit, 16-bit, 8-bit)
   - Access Count RWCS[CNT] → 0x0000 or 0x0001 (single access)

NOTE

Access count RWCS[CNT] of 0x0000 or 0x0001 performs a single access.

3. Initialize the read/write access data register (RWD) through the access method outlined in Section 36.6.9, Nexus3+ Register Access via JTAG / OnCE, using the Nexus register index of 0xA (see Table 36-19). Configure as follows:
   - Write Data → 0x*nnnn_nnnn* (write data)
4. The Nexus3+ module then arbitrates for the system bus and transfer the data value from the data buffer RWD register to the memory mapped address in the read/write access address register (RWA). When the access has completed without error (ERR = 0), Nexus3+ clears the DV bit in the RWCS register. This indicates that the device is ready for the next access.

NOTE

The DV and ERR bits within the RWCS provide read/write access status to the external development tool.

36.6.10.6.2 Block Write Access (Non-Burst Mode)

1. For a non-burst block write access, follow Steps 1, 2, and 3 outlined in Section 36.6.10.6.1, Single Write Access to initialize the registers, but using a value greater than one (0x1) for the RWCS[CNT] field.
2. The Nexus3+ module then arbitrates for the system bus and transfer the first data value from the RWD register to the memory mapped address in the read/write access address register (RWA). When the transfer has completed without error (ERR = 0), the address from the RWA register is incremented to the next word size (specified in the SZ field) and the number from the CNT field is decremented.

3. Repeat step 3 in Section 36.6.10.6.1, Single Write Access until the internal CNT value is zero (0). When this occurs, the DV bit within the RWCS is cleared to indicate the end of the block write access.

36.6.10.6.3 Block Write Access (Burst Mode)

1. For a burst block write access, follow Steps 1 and 2 outlined in Section 36.6.10.6.1, Single Write Access to initialize the registers, using a value of four (double-words) for the CNT field and a RWCS[SZ] field indicating 64-bit access.
2. Initialize the burst data buffer (read/write access data register) through the access method outlined in Section 36.6.9, Nexus3+ Register Access via JTAG / OnCE, using the Nexus register Index of 0xA (see Table 36-19).
3. Repeat step 2 until all double-word values are written to the buffer.

NOTE

The data values must be shifted in 32-bits at a time LSB first (that is, double-word write = two word writes to the RWD).

4. The Nexus module then arbitrates for the system bus and transfer the burst data values from the data buffer to the system bus beginning from the memory mapped address in the read/write access address register (RWA). For each access within the burst, the address from the RWA register is incremented to the next double-word size (specified in the SZ field) modulo the length of the burst, and the number from the CNT field is decremented.
5. When the entire burst transfer has completed without error (ERR = 0), the DV bit within the RWCS is cleared to indicate the end of the block write access.

NOTE

The actual RWA value as well as the CNT field within the RWCS are not changed when executing a block write access (burst or non-burst). The original values can be read by the external development tool at any time.

36.6.10.6.4 Single Read Access

1. Initialize the read/write access address register (RWA) through the access method outlined in Section 36.6.9, Nexus3+ Register Access via JTAG / OnCE, using the Nexus register index of 0x9 (see Table 36-19). Configure as follows:
   – Read Address → 0x*nnnn_nnnn* (read address)
2. Initialize the read/write access control/status register (RWCS) through the access method outlined in Section 36.6.9, Nexus3+ Register Access via JTAG / OnCE, using the Nexus register index of 0x7 (see Table 36-19). Configure the bits as follows:
   – Access Control RWCS[AC] → 0b1 (to indicate start access)
   – Map Select RWCS[MAP] → 0b000 (primary memory map)
   – Access Priority RWCS[PR] → 0b00 (lowest priority)
   – Read/Write RWCS[RW] → 0b0 (read access)
   – Word Size RWCS[SZ] → 0b0xx (32-bit, 16-bit, 8-bit)

– Access Count RWCS[CNT] → 0x0000 or 0x0001 (single access)

NOTE

Access Count (CNT) of 0x0000 or 0x0001 performs a single access.

3. The Nexus3+ module then arbitrates for the system bus and the read data is transferred from the system bus to the RWD register. When the transfer is completed without error (ERR = 0), Nexus sets the DV bit in the RWCS register. This indicates that the device is ready for the next access.
4. The data can then be read from the read/write access data register (RWD) through the access method outlined in Section 36.6.9, Nexus3+ Register Access via JTAG / OnCE, using the Nexus register index of 0xA (see Table 36-19).

NOTE

The DV and ERR bits within the RWCS provide Read/Write Access status to the external development tool.

36.6.10.6.5 Block Read Access (Non-Burst Mode)

1. For a non-burst block read access, follow Steps 1 and 2 outlined in Section 36.6.10.6.4, Single Read Access to initialize the registers, but using a value greater than one (0x1) for the CNT field in the RWCS register.
2. The Nexus3+ module then arbitrates for the system bus and the read data is transferred from the system bus to the RWD register. When the transfer has completed without error (ERR = 0), the address from the RWA register is incremented to the next word size (specified in the SZ field) and the number from the CNT field is decremented.
3. The data can then be read from the read/write access data register (RWD) through the access method outlined in Section 36.6.9, Nexus3+ Register Access via JTAG / OnCE, using the Nexus register index of 0xA (see Table 36-19).
4. Repeat steps 3 and 4 in Section 36.6.10.6.4, Single Read Access until the CNT value is zero (0). When this occurs, the DV bit within the RWCS is set to indicate the end of the block read access.

36.6.10.6.6 Block Read Access (Burst Mode)

1. For a burst block read access, follow Steps 1 and 2 outlined in Section 36.6.10.6.4, Single Read Access to initialize the registers, using a value of four (double-words) for the CNT field and an RWCS[SZ] field indicating 64-bit access.
2. The Nexus3+ module then arbitrates for the system bus and the burst read data is transferred from the system bus to the data buffer (RWD register). For each access within the burst, the address from the RWA register is incremented to the next double-word (specified in the SZ field) and the number from the CNT field is decremented.
3. When the entire burst transfer has completed without error (ERR = 0), the DV bit within the RWCS is set to indicate the end of the block read access.
4. The data can then be read from the burst data buffer (read/write access data register) through the access method outlined in Section 36.6.9, Nexus3+ Register Access via JTAG / OnCE, using the Nexus register index of 0xA (see Table 36-19).
5. Repeat step 3 until all double-word values are read from the buffer.

NOTE

The data values must be shifted out 32-bits at a time LSB first (that is, double-word read = two word reads from the RWD).

NOTE

The actual RWA value as well as the CNT field within the RWCS are not changed when executing a block read access (burst or non-burst). The original values can be read by the external development tool at any time.

36.6.10.6.7 Error Handling

The Nexus3+ module handles various error conditions as follows:

System Bus Read/Write Error

All address and data errors that occur on read/write accesses to the e200z6 system bus returns a transfer error. If this occurs:

1. The access is terminated without re-trying (AC bit is cleared).
2. The ERR bit in the RWCS register is set.
3. The error message is sent (TCODE = 8) indicating read/write error.

Access Termination

The following cases are defined for sequences of the read/write protocol that differ from those described in the above sections:

1. If the AC bit in the RWCS register is set to start read/write accesses and invalid values are loaded into the RWD and/or RWA, then a system bus access error may occur. This is handled as described above.
2. If a block access is in progress (all cycles not completed), and the RWCS register is written, then the original block access is terminated at the boundary of the nearest completed access.
   a) If the RWCS is written with the AC bit set, the next read/write access begins and the RWD can be written to/ read from.
   b) If the RWCS is written with the AC bit cleared, the read/write access is terminated at the nearest completed access. This method can be used to break (early terminate) block accesses.

36.6.10.6.8 Read/Write Access Error Message

The read/write access error message is sent out when an system bus access error (read or write) has occurred.

Error information is messaged out in the following format:

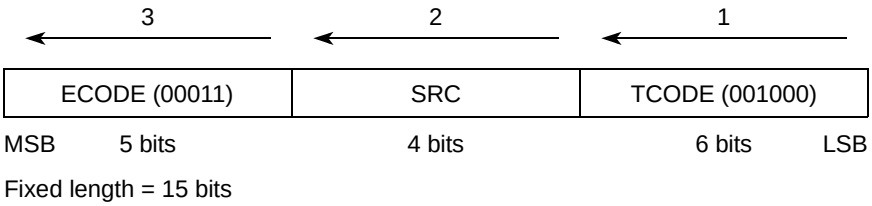


Figure 36-51. Error Message Format

36.6.10.7 Examples

The following are examples of program trace and data trace messages.

Table 36-38 illustrates an example indirect branch message with an 12 MDO / 2 MSEO configuration.

Note that T0 and S0 are the least significant bits where:

- T*x* = TCODE number (fixed)
- S*x* = Source processor (fixed)
- I*x* = Number of instructions (variable)
- A*x* = Unique portion of the address (variable)

Table 36-38. Indirect Branch Message Example (12 MDO / 2 MSEO)

Clock	MDO[11:0]												MSEO[1:0]		State
	11	10	9	8	7	6	5	4	3	2	1	0			
0	X	X	X	X	X	X	X	X	X	X	X	X	1	1	Idle (or end of last message)
1	I1	I0	S3	S2	S1	S0	T5	T4	T3	T2	T1	T0	0	0	Start Message
2	0	0	0	0	0	0	0	0	I5	I4	I3	I2	0	1	End Packet
3	0	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	1	1	End Packet/End Message
4	X	X	S3	S2	S1	S0	T5	T4	T3	T2	T1	T0	0	0	Start of Next Message

Table 36-39 illustrates an example of direct branch message with 12 MDO / 2 MSEO.

Note that T0 and I0 are the least significant bits where:

- T*x* = TCODE number (fixed)
- S*x* = Source processor (fixed)
- I*x* = Number of instructions (variable)

Table 36-39. Direct Branch Message Example (12 MDO / 2 MSEO)

Clock	MDO[11:0]												MSEO[1:0]		State
	11	10	9	8	7	6	5	4	3	2	1	0			
0	X	X	X	X	X	X	X	X	X	X	X	X	1	1	Idle (or end of last message)
1	I1	I0	S3	S2	S1	S0	T5	T4	T3	T2	T1	T0	0	0	Start Message

Table 36-39. Direct Branch Message Example (12 MDO / 2 MSEO) (continued)

Clock	MDO[11:0]												MSEO[1:0]		State
	11	10	9	8	7	6	5	4	3	2	1	0			
2	0	0	0	0	0	0	0	0	0	0	I3	I2	1	1	End Packet/End Message
3	X	X	X	X	S1	S0	T5	T4	T3	T2	T1	T0	0	0	Start of Next Message

Table 36-40 is an example data write message with 12 MDO / 2 MSEO configuration

Note that T0, A0, D0 are the least significant bits where:

- T*x* = TCODE number (fixed)
- S*x* = Source processor (fixed)
- Z*x* = Data size (fixed)
- A*x* = Unique portion of the address (variable)
- D*x* = Write data (variable - 8, 16 or 32-bit)

Table 36-40. Direct Write Message Example (12 MDO / 2 MSEO)

Clock	MDO[11:0]												MSEO[1:0]		State
	11	10	9	8	7	6	5	4	3	2	1	0			
0	X	X	X	X	X	X	X	X	X	X	X	X	1	1	Idle (or end of last message)
1	Z1	Z0	S3	S2	S1	S0	T5	T4	T3	T2	T1	T0	0	0	Start Message
2	0	0	0	0	0	0	0	A3	A2	A1	A0	Z2	0	1	End Packet
3	X	X	X	X	D7	D6	D5	D4	D3	D2	D1	D0	1	1	End Packet/End Message

36.6.10.8 IEEE 1149.1 (JTAG) RD/WR Sequences

This section contains example JTAG/OnCE sequences used to access resources.

36.6.10.8.1 JTAG Sequence for Accessing Internal Nexus Registers

Table 36-41. Accessing Internal Nexus3 Registers via JTAG/OnCE

Step #	TMS Pin	Description
1	1	IDLE → SELECT-DR_SCAN
2	0	SELECT-DR_SCAN → CAPTURE-DR (Nexus command register value loaded in shifter)
3	0	CAPTURE-DR → SHIFT-DR
4	0	(7) TCK clocks issued to shift in direction (rd/wr) bit and first 6 bits of Nexus reg. addr.
5	1	SHIFT-DR → EXIT1-DR (7th bit of Nexus reg. shifted in)
6	1	EXIT1-DR → UPDATE-DR (Nexus shifter is transferred to Nexus command register)
7	1	UPDATE-DR → SELECT-DR_SCAN
8	0	SELECT-DR_SCAN → CAPTURE-DR (Register value is transferred to Nexus shifter)

Table 36-41. Accessing Internal Nexus3 Registers via JTAG/OnCE (continued)

Step #	TMS Pin	Description
9	0	CAPTURE-DR → SHIFT-DR
10	0	(31) TCK clocks issued to transfer register value to TDO pin while shifting in TDI value
11	1	SHIFT-DR → EXIT1-DR (MSB of value is shifted in/out of shifter)
12	1	EXIT1-DR → UPDATE -DR (if access is write, shifter is transferred to register)
13	0	UPDATE-DR → RUN-TEST/IDLE (transfer complete - Nexus controller to reg. select state)

36.6.10.8.2 JTAG Sequence for Read Access of Memory-Mapped Resources

Table 36-42. Accessing Memory-Mapped Resources (Reads)

Step #	TCLK clocks	Description
1	13	Nexus Command = write to read/write access address register (RWA)
2	37	Write RWA (initialize starting read address—data input on TDI)
3	13	Nexus Command = write to read/write control/status register (RWCS)
4	37	Write RWCS (initialize read access mode and CNT value—data input on TDI)
5	13	Nexus Command = read read/write access data register (RWD)
6	37	Read RWD (data output on TDO)
7	—	If CNT > 0, go back to Step #5

36.6.10.8.3 JTAG Sequence for Write Access of Memory-Mapped Resources

Table 36-43. Accessing Memory-Mapped Resources (Writes)

Step #	TCLK clocks	Description
1	13	Nexus Command = write to read/write access control/status register (RWCS)
2	37	Write RWCS (initialize write access mode and CNT value—data input on TDI)
3	13	Nexus Command = write to read/write address register (RWA)
4	37	Write RWA (initialize starting write address—data input on TDI)
5	13	Nexus Command = read read/write access data register (RWD)
6	37	Write RWD (data output on TDO)
7	—	If CNT > 0, go back to Step #5

36.7 e200z0 Class 2+ Nexus Module (Nexus2+)

The Nexus2+ module provides real-time development capabilities for the device core in compliance with the IEEE-ISTO Nexus 5001-2003 standard. This module provides development support capabilities without requiring the use of address and data pins for internal visibility.

A portion of the pin interface (the JTAG port) is also shared with the OnCE / Nexus 1 unit. The IEEE-ISTO 5001-2003 standard defines an extensible auxiliary port that is used in conjunction with the JTAG port in e200z0 processors.

36.7.1 Nexus2+ Introduction

This section defines the auxiliary pin functions, transfer protocols and standard development features of the Nexus2+ module. The development features supported are Program trace, watchpoint messaging, ownership trace, and read/write access via the JTAG interface. The Nexus2+ module also supports two class four features: Watchpoint Triggering and Processor Overrun Control.

NOTE

Throughout this section references are made to the auxiliary port and its specific signals, such as MCKO, $\overline{\text{MSEO}}$[1:0], MDO[11:0], and others. In actual use, the device NPC module arbitrates the access of the single auxiliary port. To simplify the description of the function of the Nexus2+ module, the interaction of the NPC is omitted and the behavior described as if the module has its own dedicated auxiliary port.

36.7.2 Nexus2+ Block Diagram

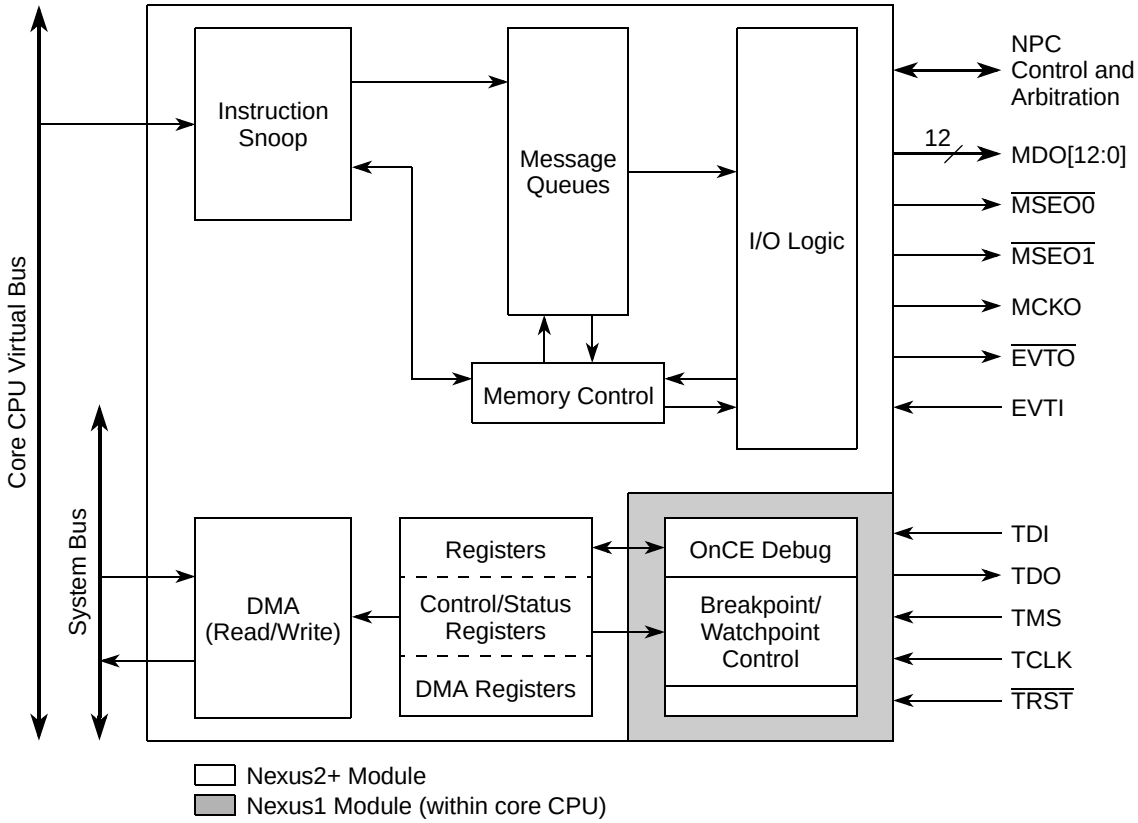


Figure 36-52. e200z0 Nexus2+ Functional Block Diagram

36.7.3 Nexus2+ Features

The Nexus2+ module is compliant with Class 2 of the IEEE-ISTO 5001-2003 standard, with additional Class 3 and 4 features available. The following features are implemented:

- Program trace via branch trace messaging (BTM). Branch trace messaging displays program flow discontinuities (direct and indirect branches, exceptions, etc.), allowing the development tool to interpolate what transpires between the discontinuities. Thus static code may be traced.
- Ownership trace via ownership trace messaging (OTM). OTM facilitates ownership trace by providing visibility of which process ID or operating system task is activated. An ownership trace message is transmitted when a new process/task is activated, allowing the development tool to trace ownership flow.
- Run-time access to embedded processor registers and memory map via the JTAG port. This allows for enhanced download/upload capabilities.
- Watchpoint messaging via the auxiliary pins.
- Watchpoint trigger enable of program and/or data trace messaging.
- Higher speed data input/output via the auxiliary port.
- Registers for program trace, ownership trace and watchpoint trigger.
- All features controllable and configurable via the JTAG port.

36.7.4 Enabling Nexus2+ Operation

The Nexus module is enabled by loading a single instruction (ACCESS_AUX_TAP_Z0, as shown in Table 35-2) into the JTAGC instruction register (IR), and then loading the corresponding OnCE OCMD register with the NEXUS2_ACCESS instruction (refer to Table 36-2). For the e200z0 Class 2+ Nexus module, the OCMD value is 0b00_0111_1100. Once enabled, the module is ready to accept control input via the JTAG pins. See Section 36.4.1.1, Enabling Nexus Clients for TAP Access for more information.

Enabling the Nexus 2+ module automatically enables the generation of Debug Status Messages.

The Nexus module is disabled when the JTAG state machine reaches the test-logic-reset state. This state can be reached by the negation of the JCOMP pin or by cycling through the state machine using the TMS pin. The Nexus module also is disabled if a power-on-reset (POR) event occurs. If the Nexus2+ module is disabled, no trace output is provided, and the module disables (drives inactive) auxiliary port output pins MDO[11:0], MSEO[1:0], MCKO. Nexus registers are not available for reads or writes.

36.7.5 TCODEs Supported by Nexus2+

The Nexus2+ pins allow for flexible transfer operations via public messages. A TCODE defines the transfer format, the number and/or size of the packets to be transferred, and the purpose of each packet. The IEEE-ISTO 5001-2003 standard defines a set of public messages. The Nexus2+ module supports the public TCODEs seen in Table 36-44. Each message contains multiple packets transmitted in the order shown in the table.

Table 36-44. Public TCODEs Supported by Nexus2+

Message Name	Packet Size (bits)		Packet Name	Packet Type	Packet Description
	Min	Max			
Debug Status	6	6	TCODE	Fixed	TCODE number = 0 (0x00)
	4	4	SRC	Fixed	source processor identifier
	8	8	STATUS	Fixed	Debug status register (DS[31:24])
Ownership Trace Message	6	6	TCODE	Fixed	TCODE number = 2 (0x02)
	4	4	SRC	Fixed	source processor identifier
	32	32	PROCESS	Fixed	Task/Process ID tag
Program Trace — Direct Branch Message[1]	6	6	TCODE	Fixed	TCODE number = 3 (0x03)
	4	4	SRC	Fixed	source processor identifier
	1	8	I-CNT	Variable	# sequential instructions executed since last taken branch
Program Trace — Indirect Branch Message[1]	6	6	TCODE	Fixed	TCODE number = 4 (0x04)
	4	4	SRC	Fixed	source processor identifier
	1	8	I-CNT	Variable	# sequential instructions executed since last taken branch
	1	32	U-ADDR	Variable	unique part of target address for taken branches/exceptions

Table 36-44. Public TCODEs Supported by Nexus2+ (continued)

Message Name	Packet Size (bits)		Packet Name	Packet Type	Packet Description
	Min	Max			
Error Message	6	6	TCODE	Fixed	TCODE number = 8 (0x08)
	4	4	SRC	Fixed	source processor identifier
	5	5	ECODE	Fixed	error code
Program Trace — Direct Branch Message w/ Sync[1]	6	6	TCODE	Fixed	TCODE number = 11 (0x0B)
	4	4	SRC	Fixed	source processor identifier
	1	8	I-CNT	Variable	# sequential instructions executed since last taken branch
	1	32	F-ADDR	Variable	full target address (leading zeros truncated)
Program Trace — Indirect Branch Message w/ Sync[1]	6	6	TCODE	Fixed	TCODE number = 12 (0x0C)
	4	4	SRC	Fixed	source processor identifier
	1	8	I-CNT	Variable	# sequential instructions executed since last taken branch
	1	32	F-ADDR	Variable	full target address (leading zeros truncated)
Watchpoint Message	6	6	TCODE	Fixed	TCODE number = 15 (0x0F)
	4	4	SRC	Fixed	source processor identifier
	4	4	WPHIT	Fixed	# indicating watchpoint sources
Resource Full Message	6	6	TCODE	Fixed	TCODE number = 27 (0x1B)
	4	4	SRC	Fixed	source processor identifier
	4	4	RCODE	Fixed	resource code (Refer to RCODE values in Table 36-46) - indicates which resource is the cause of this message
	1	32	RDATA	Variable	branch / predicate instruction history (see Section 36.7.9.3.1, Branch Trace Messaging (BTM))
Program Trace — Indirect Branch History Message	6	6	TCODE	Fixed	TCODE number = 28 (0x1C) (see footnote 1 below)
	4	4	SRC	Fixed	source processor identifier
	1	8	I-CNT	Variable	# sequential instructions executed since last taken branch
	1	32	U-ADDR	Variable	unique part of target address for taken branches/exceptions
	1	32	HIST	Variable	branch / predicate instruction history (see Section 36.7.9.3.1, Branch Trace Messaging (BTM))
Program Trace — Indirect Branch History Message w/ Sync	6	6	TCODE	Fixed	TCODE number = 29 (0x1D)[1]
	4	4	SRC	Fixed	source processor identifier
	1	8	I-CNT	Variable	# sequential instructions executed since last taken branch
	1	32	F-ADDR	Variable	full target address (leading zero (0) truncated)
	1	32	HIST	Variable	branch / predicate instruction history (see Section 36.7.9.3.1, Branch Trace Messaging (BTM))

Table 36-44. Public TCODEs Supported by Nexus2+ (continued)

Message Name	Packet Size (bits)		Packet Name	Packet Type	Packet Description
	Min	Max			
Program Trace — Program Correlation Message	6	6	TCODE	Fixed	TCODE number = 33 (0x21)
	4	4	SRC	Fixed	source processor identifier
	4	4	EVCODE	Fixed	event correlated w/ program flow (Refer to Table 36-47)
	1	8	I-CNT	Variable	# sequential instructions executed since last taken branch
	1	32	HIST	Variable	branch / predicate instruction history (see Section 36.7.9.3.1, Branch Trace Messaging (BTM))

[1] The user can select between the two types of program trace. The advantages for each are discussed in Section 36.7.9.3.1, Branch Trace Messaging (BTM). If the branch history method is selected, the shaded TCODEs above are not messaged out.

Table 36-45 shows the error code encodings used when reporting an error via the Nexus2+ Error Message.

Table 36-45. Error Code Encoding (TCODE = 8)

Error Code (ECODE)	Description
00000	Ownership trace overrun
00001	Program trace overrun
00010	Reserved
00011	Read/write access error
00101	Invalid access opcode (Nexus register unimplemented)
00110	Watchpoint overrun
00111	Program trace and ownership trace overrun
01000	(Program trace or ownership trace) and watchpoint overrun
01001–0111	Reserved
11000	BTM lost due to collision w/ higher priority message
11001–11111	Reserved

Table 36-46 shows the encodings used for resource codes for certain messages.

Table 36-46. RCODE values (TCODE = 27)

Resource Code (RCODE)	Description	Resource Data (RDATA)
0000	Program Trace Instruction Counter overflow (reached 255 and was reset)	0xFF
0001	Program Trace, Branch / Predicate Instruction History. This type of packet is terminated by a stop bit set to 1 after the last history bit.	Branch History. This type of packet is terminated by a stop bit set to a 1 after the last history bit.

Table 36-47 shows the event code encodings used for certain messages.

Table 36-47. Event Code Encoding (TCODE = 33)

Event Code	Description
0000	Entry into Debug Mode
0001	Entry into Low Power Mode (CPU only)[1]
0010–0011	Reserved for future functionality
0100	Disabling Program Trace
0101–1111	Reserved for future functionality

1 The device enters Low Power Mode when the Nexus stall mode is enabled (NEXUS2_DC1[OVC] = 0b011) and a trace message is in danger of over-flowing the Nexus queue.

36.7.6 Nexus2+ Memory Map

This section describes the Nexus2+ programmer's model. Nexus2+ registers are accessed using the JTAG/OnCE port in compliance with IEEE 1149.1. See Section 36.7.8, Nexus2+ Register Access via JTAG / OnCE for details on Nexus2+ register access.

NOTE

Nexus2+ registers and output signals are numbered using bit 0 as the least significant bit. This bit ordering is consistent with the ordering defined by the IEEE-ISTO 5001 standard.

Table 36-48 details the register map for the Nexus2+ module.

Table 36-48. Nexus2+ Memory Map

Index	Register Description	Read Address[1]	Write Address[1]	Section/Page
0x02	e200z0 Development Control1 (PPC_DC1)	0x04	0x05	36.7.7.1/36-76
0x03	e200z0 Development Control2 (PPC_DC2)	0x06	0x07	36.7.7.1/36-76
0x04	e200z0 Development Status (PPC_DS)	0x08	—	36.7.7.2/36-78
0x07	e200z0 Read/Write Access Control/Status (Nexus2_RWCS)	0x0E	0x0F	36.7.7.3/36-78
0x09	e200z0 Read/Write Access Address (Nexus2_RWA)	0x12	0x13	36.7.7.4/36-80
0x0A	e200z0 Read/Write Access Data (Nexus2_RWD)	0x14	0x15	36.7.7.5/36-80
0x0B	e200z0 Watchpoint Trigger (PPC_WT)	0x16	0x17	36.7.7.6/36-81
0xC – 0x3F	Reserved			

1 See Section 36.5.5.2.3, NPC IEEE 1149.1-2001 (JTAG) TAP, for a description of the read and write address usage for the e200z6 and e200z0 Nexus Control/Status registers.

36.7.7 Nexus2+ Register Definition

36.7.7.1 Development Control Register 1, 2 (DC1, DC2)

The development control registers are used to control the basic development features of the Nexus2+ module. Development control register 1 is shown in Figure 36-53 and its fields are described in Table 36-49.

Nexus Reg: 0x2 Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	OPC	MCK_DIV		EOC		0	PTM	WEN	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R									OVC			EIC		TM		
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 36-53. Development Control Register 1 (DC1)

Table 36-49. DC1 Field Descriptions

Field	Description
OPC[1]	Output port mode control. 0 Reduced-port mode configuration (not available on the PXN20) 1 Full-port mode configuration (12 MDO pins)
MCK_DIV [1:0][1]	MCKO clock divide ratio (see note below). 00 MCKO is 1x processor clock freq. 01 MCKO is 1/2x processor clock freq. 10 MCKO is 1/4x processor clock freq. 11 MCKO is 1/8x processor clock freq.
EOC[1:0]	$\overline{\text{EVTO}}$ control. 00 $\overline{\text{EVTO}}$ upon occurrence of watchpoints (configured in DC2) 01 $\overline{\text{EVTO}}$ upon entry into debug mode 10 $\overline{\text{EVTO}}$ upon timestamping event 11 Reserved
PTM	Program trace method. 0 Program trace uses traditional branch messages 1 Program trace uses branch history messages
WEN	Watchpoint trace enable. 0 Watchpoint Messaging disabled 1 Watchpoint Messaging enabled

Table 36-49. DC1 Field Descriptions (continued)

Field	Description
OVC[2:0]	Overrun control. 000 Generate overrun messages 001–010 Reserved 011 Delay processor for BTM / DTM / OTM overruns 1XX Reserved
EIC[1:0]	$\overline{\text{EVTI}}$ control. 00 $\overline{\text{EVTI}}$ is used for synchronization (program trace/ data trace) 01 $\overline{\text{EVTI}}$ is used for debug request 1X Reserved
TM[2:0]	Trace mode. Any or all of the TM bits may set, enabling one or more traces. 000 No trace 1XX Program trace enabled X1X Data trace enabled XX1 Ownership trace enabled

1 The output port mode control bit (OPC) and MCKO divide bits (MCK_DIV) are shown for clarity. These functions are controlled globally by the NPC port control register (PCR). These bits are writable in the PCR but have no effect.

Development control register 2 is shown in Figure 36-54 and its fields are described in Table 36-50.

Nexus Reg: 0x3 Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	EWC								0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 36-54. Development Control Register 2 (DC2)

Table 36-50. DC2 Field Descriptions

Field	Description
EWC[7:0]	$\overline{\text{EVTO}}$ watchpoint configuration. Any or all of the bits in EWC may be set to configure the $\overline{\text{EVTO}}$ watchpoint. 00000000No Watchpoints trigger $\overline{\text{EVTO}}$ 1XXXXXXXWatchpoint #0 (IAC1 from Nexus1) triggers $\overline{\text{EVTO}}$ X1XXXXXXWatchpoint #1 (IAC2 from Nexus1) triggers $\overline{\text{EVTO}}$ XX1XXXXXWatchpoint #2 (IAC3 from Nexus1) triggers $\overline{\text{EVTO}}$ XXX1XXXXWatchpoint #3 (IAC4 from Nexus1) triggers $\overline{\text{EVTO}}$ XXXX1XXXWatchpoint #4 (DAC1 from Nexus1) triggers $\overline{\text{EVTO}}$ XXXXX1XXWatchpoint #5 (DAC2 from Nexus1) triggers $\overline{\text{EVTO}}$

NOTE

The EOC bits in DC1 must be programmed to trigger $\overline{\text{EVTO}}$ on watchpoint occurrence for the EWC bits to have any effect.

36.7.7.2 Development Status Register (DS)

The development status register is used to report system debug status. When debug mode is entered or exited, or an e200z0-defined low power mode is entered, a debug status message is transmitted with DS[31:24]. The external tool can read this register at any time.

Nexus Reg: 0x4 Access: User read only

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	DBG	0	0	0	LPC		CHK	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 36-55. Development Status Register (DS)

Table 36-51. DS Field Descriptions

Field	Description
DBG	e200z0 CPU debug mode status. 0 CPU not in debug mode 1 CPU in debug mode (**jd_debug_b** signal asserted)
LPC[1:0]	e200z0 CPU low power mode status. 00 Normal (run) mode 01 CPU in halted state 10 CPU in stopped state 11 Reserved
CHK	e200z0 CPU checkstop status. 0 CPU not in checkstop state 1 CPU in checkstop state

36.7.7.3 Read/Write Access Control/Status (RWCS)

The read write access control/status register provides control for read/write access. Read/write access provides DMA-like access to memory-mapped resources on the system bus either while the processor is halted, or during runtime. The RWCS register also provides read/write access status information as shown in Table 36-52.

Nexus Reg: 0x7 Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	AC	RW	SZ			MAP			PR		BST	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	CNT														ERR	DV
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 36-56. Read/Write Access Control/Status Register (RWCS)

Table 36-52. RWCS Field Description

Field	Description
AC	Access control. 0 End access 1 Start access
RW	Read/write select. 0 Read access 1 Write access
SZ[2:0]	Word size. 000 8-bit (byte) 001 16-bit (half-word) 010 32-bit (word) 011 Reserved 100–111 Reserved (default to word)
MAP[2:0]	MAP select. 000 Primary memory map 001-111 Reserved
PR[1:0]	Read/write access priority. 00 Lowest access priority 01 Reserved (default to lowest priority) 10 Reserved (default to lowest priority) 11 Highest access priority
BST	Burst control. 0 Module accesses are single bus cycle at a time. 1 Module accesses are performed as burst operation.
CNT[13:0]	Access control count. Number of accesses of word size SZ
ERR	Read/write access error. See Table 36-53.
DV	Read/write access data valid. See Table 36-53.

Table 36-53 details the status bit encodings.

Table 36-53. Read/Write Access Status Bit Encoding

Read Action	Write Action	ERR	DV
Read access has not completed	Write access completed without error	0	0
Read access error has occurred	Write access error has occurred	1	0
Read access completed without error	Write access has not completed	0	1
Not allowed	Not allowed	1	1

36.7.7.4 Read/Write Access Address (RWA)

The read/write access address register provides the system bus address to be accessed when initiating a read or a write access.

Nexus Reg: 0x9 Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	Read/Write Address																															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 36-57. Read/Write Access Address Register (RWA)

36.7.7.5 Read/Write Access Data (RWD)

The read/write access data register provides the data to/from system bus memory-mapped locations when initiating a read or a write access.

Nexus Reg: 0xA Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	Read/Write Data																															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 36-58. Read/Write Access Data Register (RWD)

Table 36-54 shows the proper placement of data into the RWD. The “X” in the RWD column indicate byte lanes with valid data.

Table 36-54. RWD Data Placement for Transfers

Transfer Size and byte offset	RWA[2:0	RWCS[SZ]	RWD			
			31:24	23:16	15:8	7:0
Byte	XXX	000	—	—	—	X
Half Word	XX0	001	—	—	X	X
Word	X00	010	X	X	X	X

Table 36-55 shows the mapping of RWD bytes to byte lanes of the AHB read and write data buses.

Table 36-55. RWD data placement for Transfers

Transfer Size and byte offset	RWA[2:0]	RWD			
		31:24	23:16	15:8	7:0
Byte @000	000	—	—	—	AHB[7:0]
Byte @001	001	—	—	—	AHB[15:8]
Byte @010	010	—	—	—	AHB[23:16]
Byte @011	011	—	—	—	AHB[31:24]
Half@000	000	—	—	AHB[15:8]	AHB[7:0]
Half@010	010	-	-	AHB[31:24]	AHB[23:16]
Word@000	000	AHB[31:24]	AHB[23:16]	AHB[15:8]	AHB[7:0]

36.7.7.6 Watchpoint Trigger Register (WT)

The watchpoint trigger register allows the watch points defined within the e200z0 Nexus1 logic to trigger actions. These watch points can control program trace enable and disable. The WT bits can be used to produce an address related ‘window’ for triggering trace messages.

Nexus Reg: 0xB — Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	PTS				PTE		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 36-59. Watchpoint Trigger Register (WT)

Table 36-56 details the watchpoint trigger register fields.

Table 36-56. WT Field Descriptions

Field	Description
PTS[2:0]	Program trace start control. 000 Trigger disabled 001 Use watchpoint #0 (IAC1 from Nexus1) 010 Use watchpoint #1 (IAC2 from Nexus1) 011 Use watchpoint #2 (IAC3 from Nexus1) 100 Use watchpoint #3 (IAC4 from Nexus1) 101 Use watchpoint #4 (DAC1 from Nexus1) 110 Use watchpoint #5 (DAC2 from Nexus1) 111 Reserved
PTE[2:0]	Program trace end control. 000 Trigger disabled 001 Use watchpoint #0 (IAC1 from Nexus1) 010 Use watchpoint #1 (IAC2 from Nexus1) 011 Use watchpoint #2 (IAC3 from Nexus1) 100 Use watchpoint #3 (IAC4 from Nexus1) 101 Use watchpoint #4 (DAC1 from Nexus1) 110 Use watchpoint #5 (DAC2 from Nexus1) 111 Reserved

NOTE

The WT bits control program trace only if the TM bits in the development control register 1 (DC1) have not already been set to enable program trace.

36.7.8 Nexus2+ Register Access via JTAG / OnCE

Access to Nexus2+ register resources is enabled by loading a single instruction (ACCESS_AUX_TAP_Z0) into the JTAGC instruction register (IR), and then loading the corresponding OnCE OCMD register with the NEXUS2_ACCESS instruction (refer to Table 36-2). For the Nexus2+ module, the OCMD value is 0b00_0111_1100.

Once the ACCESS_AUX_TAP_Z0 instruction has been loaded, the JTAG/OnCE port allows tool/target communications with all Nexus2+ registers according to the register map in Table 36-48.

Reading/writing of a Nexus2+ register then requires two (2) passes through the data-scan (DR) path of the JTAG state machine (see 36.6.10.8).

1. The first pass through the DR selects the Nexus2+ register to be accessed by providing an index (see Table 36-48), and the direction (read/write). This is achieved by loading an 8-bit value into the JTAG data register (DR). This register has the following format:

(7 bits)	(1 bit)
Nexus Register Index	R/W

RESET Value: 0x00

Nexus Register Index:	Selected from values in Table 36-48
Read/Write (R/W):	0 Read 1 Write

2. The second pass through the DR then shifts the data in or out of the JTAG port, LSB first.
   a) During a read access, data is latched from the selected Nexus register when the JTAG state machine passes through the CAPTURE-DR state.
   b) During a write access, data is latched into the selected Nexus register when the JTAG state machine passes through the UPDATE-DR state.

36.7.9 Nexus2+ Functional Description

36.7.9.1 Debug Status Messages

Debug Status Messages

NOTE

Debug Status Messages (DSMs) are enabled if the Nexus module is enabled.

Debug status messages report low power mode and debug status. Entering/exiting debug mode as well as entering a low power mode triggers a debug status message. Debug status information is sent out in the following format:

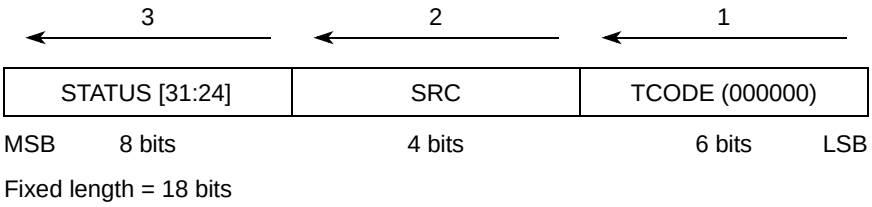


Figure 36-60. Debug Status Message Format

36.7.9.2 Ownership Trace

This section details the ownership trace features of the Nexus2+ module.

36.7.9.2.1 Overview

Ownership trace provides a macroscopic view, such as task flow reconstruction, when debugging software written in a high level (or object-oriented) language. It offers the highest level of abstraction for tracking operating system software execution. This is especially useful when the developer is not interested in debugging at lower levels.

36.7.9.2.2 Ownership Trace Messaging (OTM)

Ownership trace information is messaged via the auxiliary port using an ownership trace message (OTM). The e200z0 processor contains a Power Architecture Book E defined process ID register within the CPU.

The process ID register is updated by the operating system software to provide task/process ID information. The contents of this register are replicated on the pins of the processor and connected to Nexus. The process ID register value can be accessed using the **se_mfspr/se_mtspr** instructions. Please refer to the *e200z0 Power Architecture*™ *Core Reference Manual* for more details on the process ID register.

One condition causes an ownership trace message: When new information is updated in the OTR register or process ID register by the e200z0 processor, the data is latched within Nexus, and is messaged out via the auxiliary port, allowing development tools to trace ownership flow.

Ownership trace information is messaged out in the following format:

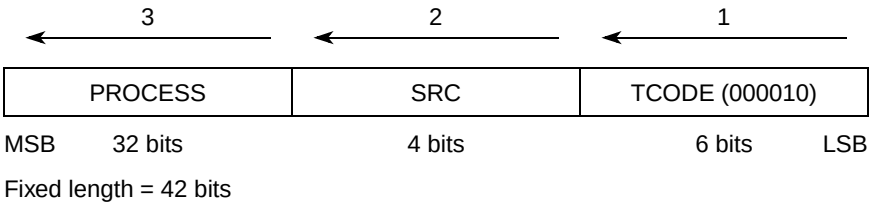


Figure 36-61. Ownership Trace Message Format

36.7.9.2.3 OTM Error Messages

An error message occurs when a new message cannot be queued due to the message queue being full. The FIFO discards incoming messages until it has completely emptied the queue. Once emptied, an error message is queued. The error encoding indicates which types of messages attempted to be queued while the FIFO was being emptied.

If only an OTM message attempts to enter the queue while it is being emptied, the error message incorporates the OTM only error encoding (00000). If both OTM and either BTM or DTM messages attempt to enter the queue, the error message incorporates the OTM and (program or data) trace error encoding (00111). If a watchpoint also attempts to be queued while the FIFO is being emptied, then the error message incorporates error encoding (01000).

NOTE

The OVC bits within the DC1 register can be set to delay the CPU in order to alleviate (but not eliminate) potential overrun situations.

Error information is messaged out in the following format (see Table 36-45):

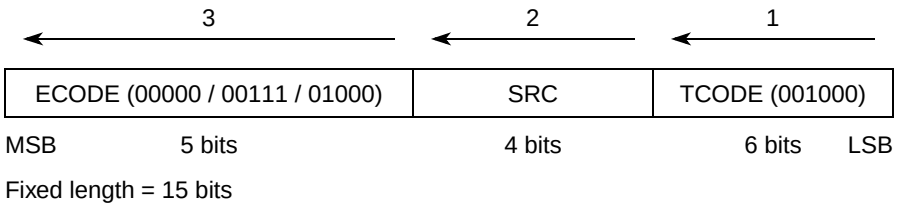


Figure 36-62. Error Message Format

36.7.9.2.4 OTM Flow

Ownership trace messages are generated when the operating system writes to the e200z0 process ID register or the memory mapped ownership trace register.

The following flow describes the OTM process:

1. The process ID register is a system control register. It is internal to the e200z0 processor and can be accessed by using PPC instructions **se_mtspr** and **se_mfspr**. The contents of this register are replicated on the pins of the processor and connected to Nexus.
2. OTR/process ID register reads do not cause ownership trace messages to be transmitted by the Nexus2+ module.
3. If the periodic OTM message counter expires (after 255 queued messages without an OTM), an OTM is sent using the latched data from the previous OTM or process ID register write.

36.7.9.3 Program Trace

This section details the program trace mechanism supported by Nexus2+ for the e200z0 processor. Program trace is implemented via branch trace messaging (BTM) as per the Class 3 IEEE-ISTO 5001-2003 standard definition. Branch trace messaging for e200z0 processors is accomplished by snooping the e200z0 address bus, attribute signals, and CPU status.

36.7.9.3.1 Branch Trace Messaging (BTM)

Traditional branch trace messaging facilitates program trace by providing the following types of information:

- Messaging for taken direct branches includes how many sequential instructions were executed since the last taken branch or exception. Direct (or indirect) branches not taken are counted as sequential instructions.
- Messaging for taken indirect branches and exceptions includes how many sequential instructions were executed since the last taken branch or exception and the unique portion of the branch target address or exception vector address.

Branch history messaging facilitates program trace by providing the following information:

- Messaging for taken indirect branches and exceptions includes how many sequential instructions were executed since the last predicate instruction, taken indirect branch, or exception, the unique portion of the branch target address or exception vector address, as well as a branch/predicate instruction history field. Each bit in the history field represents a direct branch or predicated instruction where a value of one (1) indicates taken, and a value of zero (0) indicates not taken. Certain instructions (**evsel**) generate a pair of predicate bits that are both reported as consecutive bits in the history field.

e200z0 Indirect Branch Message Instructions

Table 36-57 shows the types of instructions and events that cause indirect branch messages or branch history messages to be encoded.

Table 36-57. Indirect Branch Message Sources

Source of Indirect Branch Message	Instructions
Taken branch relative to a register value	**se_bctr, se_bctrl, se_blr, se_blrl**
System Call / Trap exceptions taken	**sc, se_sc, tw**
Return from interrupts / exceptions	**se_rfi, se_rfci, se_rfdi**
Exit from reset with Program Trace Enabled	Indirect branch with Sync, target address is initial instruction, count = 1

e200z6 Direct Branch Message Instructions

Table 36-58 shows the types of instructions that cause direct branch messages or toggle a bit in the instruction history buffer to be messaged out in a resource full message or branch history message.

Table 36-58. Direct Branch Message Sources

Source of Direct Branch Message	Instructions
Taken direct branch instructions	**se_b. se_bc, se_bl, e_b, e_bc, e_bl, e_bcl**
Instruction Synchronize	se_isync

BTM Using Branch History Messages

Traditional BTM messaging can accurately track the number of sequential instructions between branches, but cannot accurately indicate which instructions were conditionally executed, and which were not.

Branch history messaging solves this problem by providing a predicated instruction history field in each indirect branch message. Each bit in the history represents a predicated instruction or direct branch. A value of one (1) indicates the conditional instruction was executed or the direct branch was taken. A value of zero (0) indicates the conditional instruction was not executed or the direct branch was not taken. Certain instructions (**evsel**) generate a pair of predicate bits that are both reported as consecutive bits in the history field.

Branch history messages solve predicated instruction tracking and save bandwidth since only indirect branches cause messages to be queued.

BTM Using Traditional Program Trace Messages

Based on the PTM bit in the DC register (DC[PTM]), program tracing can utilize either branch history messages (DC[PTM] = 1) or traditional direct/indirect branch messages (DC[PTM] = 0).

Branch history saves bandwidth and keeps consistency between methods of program trace, yet may lose temporal order between BTM messages and other types of messages. Since direct branches are not messaged, but are instead included in the history field of the indirect branch history message, other types of messages may enter the FIFO between branch history messages. The development tool cannot determine the ordering of “events” that occurred with respect to direct branches simply by the order in which messages are sent out.

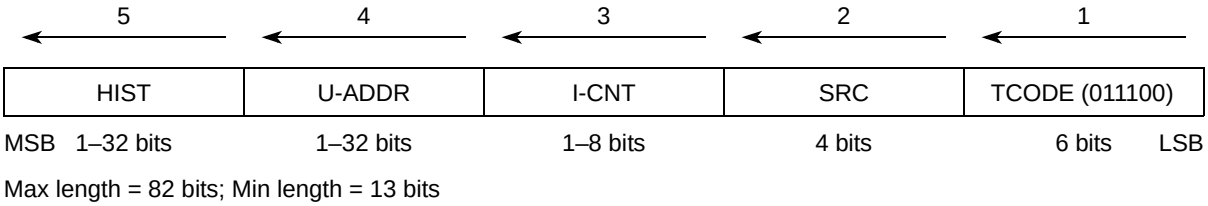
Traditional BTM messages maintain their temporal ordering because each event that can cause a message to be queued enters the FIFO in the order it occurred and is messaged out in that same order.

36.7.9.3.2 BTM Message Formats

The e200z0 Nexus2+ module supports three types of traditional BTM messages—direct, indirect, and synchronization messages. It supports two types of branch history BTM messages—indirect branch history, and indirect branch history with synchronization messages. Debug status messages and error messages are also supported.

Indirect Branch Messages (History)

Indirect branches include all taken branches whose destination is determined at run time, interrupts and exceptions. If DC[PTM] is set, indirect branch information is messaged out in the following format:

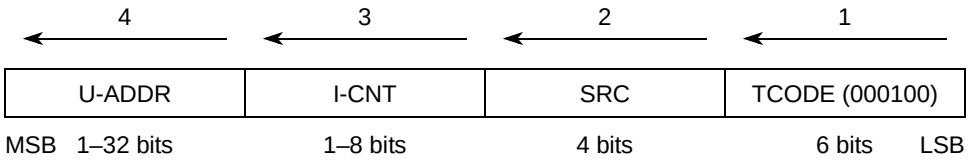


Max length = 82 bits; Min length = 13 bits

Figure 36-63. Indirect Branch Message (History) Format

Indirect Branch Messages (Traditional)

If DC[PTM] is cleared, indirect branch information is messaged out in the following format:

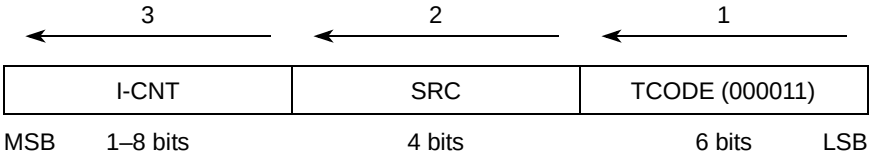


Max length = 50 bits; Min length = 12 bits

Figure 36-64. Indirect Branch Message Format

Direct Branch Messages (Traditional)

Direct branches (conditional or unconditional) are all taken branches whose destination is fixed in the instruction opcode. Direct branch information is messaged out in the following format:



Max length = 18 bits; Min length = 11 bits

Figure 36-65. Direct Branch Message Format

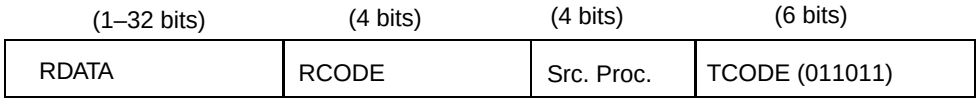
NOTE

When DC[PTM] is set, direct branch messages are not transmitted. Instead, each direct branch or predicated instruction toggles a bit in the history buffer.

Resource Full Messages

The resource full message is used in conjunction with the branch history messages. The resource full message is generated when the internal branch/predicate history buffer is full, or if the BTM Instruction sequence counter (I-CNT) overflows. If synchronization is needed at the time this message is generated, the synchronization is delayed until the next branch trace message that is not a resource full message.

The current value of the history buffer is transmitted as part of the resource full message. This information can be concatenated by the tool with the branch/predicate history information from subsequent messages to obtain the complete branch history for a message. The internal history value is reset by this message, and the I-CNT value is reset as a result of a bit being added to the history buffer.



Max length = 46 bits; Min length = 15 bits

Figure 36-66. Resource Full Message Format

Table 36-59 shows the RCODE encodings and RDATA information used for Resource Full messages.

Table 36-59. RCODE Encoding

RCODE	Description	RDATA field
0000	Program Trace Instruction counter reached 255 and was reset.	0xFF
0001	Program Trace, Branch / Predicate Instruction History full.	Branch HIstory. This type of packet is terminated by a stop bit set to 1 after the last history bit.

Program Correlation Messages

Program correlation messages are used to correlate events to the program flow that may not be associated with the instruction stream. The following events result in a PCM when program trace is enabled:

- When the CPU enters debug mode, a PCM is generated. The instruction count and history information provided by the PCM can be used to determine the last sequence of instructions executed prior to debug mode entry.
- When the CPU enters a low power mode in which instructions are no longer executed, a PCM is generated. The instruction count and history information provided by the PCM can be used to determine the last sequence of instructions executed prior to low power mode entry.
- Whenever program trace is disabled by any means, a PCM is generated. The instruction count and history information provided by the PCM can be used to determine the last sequence of instructions executed prior to disabling program trace. A second PCM is generated on this event if there has been an execution mode switch into or out of a sequence of VLE instructions. This VLE state

information allows the development tool to interpret any preceding instruction count or history information in the proper context.

- Whenever the CPU crosses a page boundary that results in an execution mode switch into or out of a sequence of VLE instructions, a PCM is generated. The PCM effectively breaks up any running instruction count and history information between the two modes of operation so that the instruction count and history information can be processed by the development tool in the proper context.
- When using program trace in history mode, when a direct branch results in an execution mode switch into or out of a sequence of VLE instructions, a PCM is generated. The PCM effectively breaks up any running history information between the two modes of operation so that the history information can be processed by the development tool in the proper context.

Program correlation is messaged out in the following format:

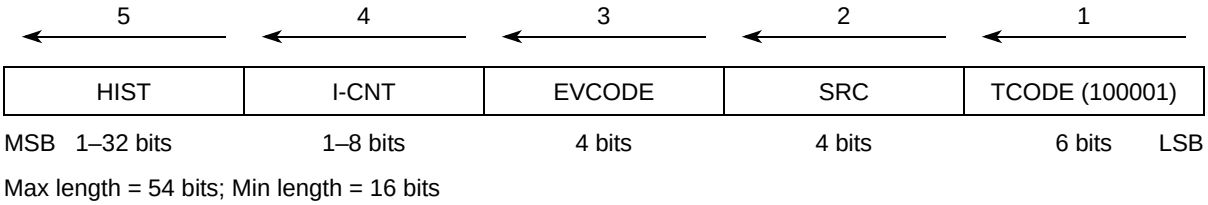


Figure 36-67. Program Correlation Message Format

BTM Overflow Error Messages

An error message occurs when a new message cannot be queued due to the message queue being full. The FIFO discards incoming messages until it has completely emptied the queue. Once emptied, an error message is queued. The error encoding indicates which types of messages attempted to be queued while the FIFO was being emptied.

If only a program trace message attempts to enter the queue while it is being emptied, the error message incorporates the program trace only error encoding (00001). If both OTM and program trace messages attempt to enter the queue, the error message incorporates the OTM and program trace error encoding (00111). If a watchpoint also attempts to be queued while the FIFO is being emptied, then the error message incorporates error encoding (01000).

NOTE

The OVC bits within the DC1 register can be set to delay the CPU in order to alleviate (but not eliminate) potential overrun situations.

Error information is messaged out in the following format:

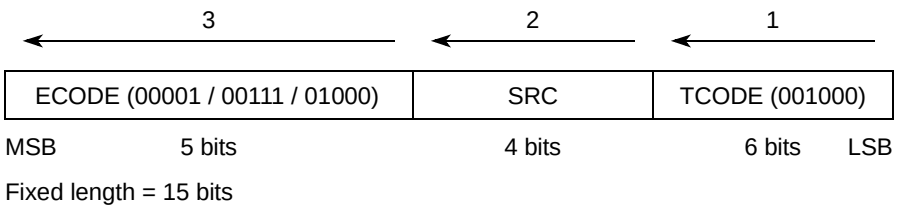


Figure 36-68. Error Message Format

Program Trace Synchronization Messages

A program trace direct/indirect branch with sync message is messaged via the auxiliary port (provided program trace is enabled) for the following conditions (see Table 36-60):

- Initial program trace message upon the first direct/indirect branch after exit from system reset or whenever program trace is enabled
- Upon direct/indirect branch after returning from a CPU low power state
- Upon direct/indirect branch after returning from debug mode
- Upon direct/indirect branch after occurrence of queue overrun (can be caused by any trace message), provided program trace is enabled
- Upon direct/indirect branch after the periodic program trace counter has expired indicating 255 *without-sync* program trace messages have occurred since the last *with-sync* message occurred
- Upon direct/indirect branch after assertion of the event in ($\overline{\text{EVTI}}$) pin if the EIC bits within the DC1 register have enabled this feature
- Upon direct/indirect branch after the sequential instruction counter has expired indicating 255 instructions have occurred between branches
- Upon direct/indirect branch after a BTM message was lost due to an attempted access to a secure memory location.
- Upon direct/indirect branch after a BTM message was lost due to a collision entering the FIFO between the BTM message and either a watchpoint message or an ownership trace message

If the Nexus2+ module is enabled at reset, a $\overline{\text{EVTI}}$ assertion initiates a program trace direct/indirect branch with sync message (if program trace is enabled) upon the first direct/indirect branch. The format for program trace direct/indirect branch with sync messages is as follows:

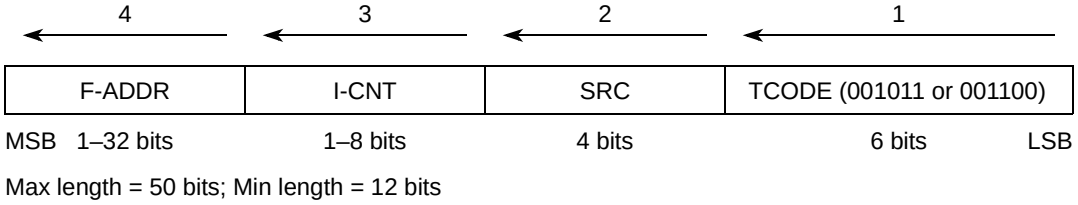


Figure 36-69. Direct/Indirect Branch with Sync Message Format

The formats for program trace direct/indirect branch with sync. messages and indirect branch history with sync. messages are as follows

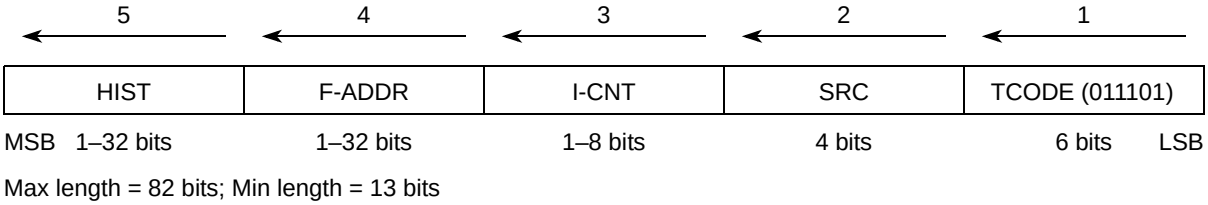


Figure 36-70. Indirect Branch History with Sync. Message Format

Exception conditions that result in program trace synchronization are summarized in Table 36-60.

Table 36-60. Program Trace Exception Summary

Exception Condition	Exception Handling
System Reset Negation	At the negation of JTAG reset (JCOMP), queue pointers, counters, state machines, and registers within the Nexus2+ module are reset. Upon the first branch out of system reset (if program trace is enabled), the first program trace message is a direct/indirect branch with sync. message.
Program Trace Enabled	The first program trace message (after program trace has been enabled) is a synchronization message.
Exit from Low Power/Debug	Upon exit from a low power mode or debug mode, the next direct/indirect branch is converted to a direct/indirect branch with sync. message.
Queue Overrun	An error message occurs when a new message cannot be queued due to the message queue being full. The FIFO discards messages until it has completely emptied the queue. Once emptied, an error message is queued. The error encoding indicates which types of messages attempted to be queued while the FIFO was being emptied. The next BTM message in the queue is a direct/indirect branch with sync. message.
Periodic Program Trace Sync.	A forced synchronization occurs periodically after 255 program trace messages have been queued. A direct/indirect branch with sync. message is queued. The periodic program trace message counter then resets.
Event In	If the Nexus module is enabled, an $\overline{\text{EVTI}}$ assertion initiates a direct/indirect branch with sync. message upon the next direct/indirect branch (if program trace is enabled and the EIC bits of the DC1 register have enabled this feature).
Sequential Instruction Count Overflow	When the sequential instruction counter reaches its maximum count (as many as 255 sequential instructions may be executed), a forced synchronization occurs. The sequential counter then resets. A program trace direct/indirect branch with sync.message is queued upon execution of the next branch.
Attempted Access to Secure Memory	For devices that implement security, any attempted branch to secure memory locations temporarily disables program trace and causes the corresponding BTM to be lost. The following direct/indirect branch queues a direct/indirect branch with sync. message. The count value within this message is inaccurate since the re-enable of program trace is not necessarily aligned on an instruction boundary.
Collision Priority	All messages have the following priority: WPM → OTM → BTM → DTM. A BTM message that attempts to enter the queue at the same time as a watchpoint message or ownership trace message is lost, and an error message is sent indicating the BTM was lost. The following direct/indirect branch queues a direct/indirect branch with sync. message. The count value within this message reflects the number of sequential instructions executed after the last successful BTM Message was generated. This count includes the branch that did not generate a message due to the collision.

36.7.9.3.3 BTM Operation

Enabling Program Trace

Both types of branch trace messaging are enabled using one of the following methods:

- Setting the TM field of the DC1 register to enable program trace (DC1[TM])
- Using the PTS field of the WT register to enable program trace on watchpoint hits (e200z0 watch points are configured within the CPU)

Relative Addressing

The relative address feature is compliant with the IEEE-ISTO 5001-2003 standard recommendations, and is designed to reduce the number of bits transmitted for addresses of indirect branch messages.

The address transmitted is relative to the target address of the instruction that triggered the previous indirect branch (or sync) message. It is generated by XOR-ing the new address with the previous address, and then using only the results up to the most significant 1 in the result. To recreate this address, an XOR of the (most-significant 0-padded) message address with the previously decoded address gives the current address.

Previous address (A1) = 0x0003_FC01, New address (A2) = 0x0003_F365

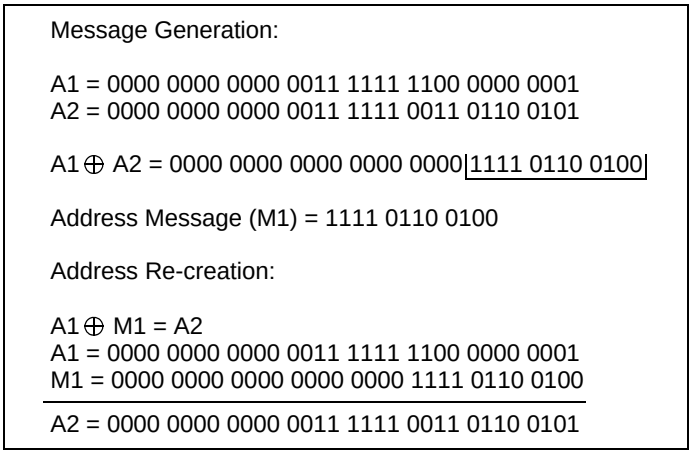


Figure 36-71. Relative Address Generation and Re-creation

Execution Mode Indication

In order for a development tool to properly interpret instruction count and history information, it must be aware of the execution mode context of that information. VLE instructions are interpreted differently from non-VLE instructions.

Program trace messages provide the execution mode status in the least significant bit of the reconstructed address field. A value of ‘0’ indicates that preceding instruction count and history information should be interpreted in a non-VLE context. A value of ‘1’ indicates that the preceding instruction count and history information should be interpreted in a VLE context. Note that when a branch results in an execution mode switch, the program trace message resulting from that branch indicates the previous execution state. The new state is not signaled until the next program trace message.

In some cases, a Program Correlation Message is generated to indicate execution mode status.

Refer to Program Correlation Messages for more information on these cases.

Branch/Predicate Instruction History (HIST)

If DC[PTM] is set, BTM messaging uses the branch history format. The branch history (HIST) packet in these messages provides a history of direct branch execution used for reconstructing the program flow. This packet is implemented as a left-shifting shift register. The register is always pre-loaded with a value

of one (1). This bit acts as a stop bit so that the development tools can determine which bit is the end of the history information. The pre-loaded bit itself is not part of the history, but is transmitted with the packet.

A value of one (1) is shifted into the history buffer on a taken branch (condition or unconditional) and on any instruction whose predicate condition executed as true. A value of zero (0) is shifted into the history buffer on any instruction whose predicate condition executed as false as well as on branches not taken. This includes indirect as well as direct branches not taken. For the **evsel** instruction, two bits are shifted in, corresponding to the low element (shifted in first) and the high element (shifted in second) conditions.

Sequential Instruction Count (I-CNT)

The I-CNT packet, is present in all BTM messages. For traditional branch messages, I-CNT represents the number of sequential instructions, or non-taken branches in between direct/indirect branch messages.

For branch history messages, I-CNT represents the number of instructions executed since the last taken/non-taken direct branch, last taken indirect branch or exception. Not taken indirect branches are considered sequential instructions and cause the instruction count to increment. I-CNT also represents the number of instructions executed since the last predicate instruction.

The sequential instruction counter overflows when its value reaches 255. The next BTM message is converted to a synchronization type message.

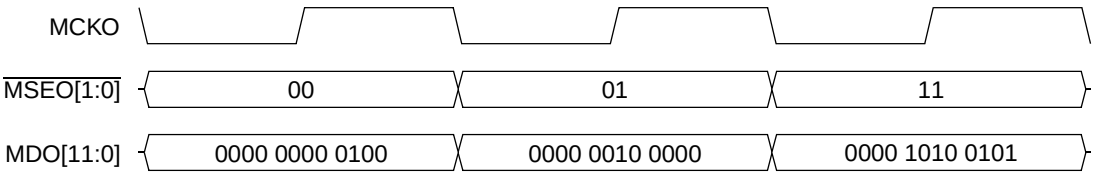
Program Trace Queueing

Nexus2+ implements a message queue. Messages that enter the queue are transmitted via the auxiliary pins in the order in which they are queued.

NOTE

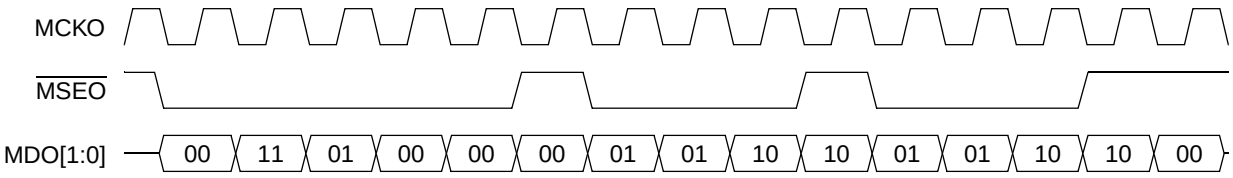
If multiple trace messages need to be queued at the same time, Watchpoint Messages have the highest priority (WPM → OTM → BTM → DTM).

36.7.9.3.4 Program Trace Timing Diagrams



TCODE = 4
Source Processor = 0b0000
Number of Sequence Instructions = 128
Relative Address = 0xA5

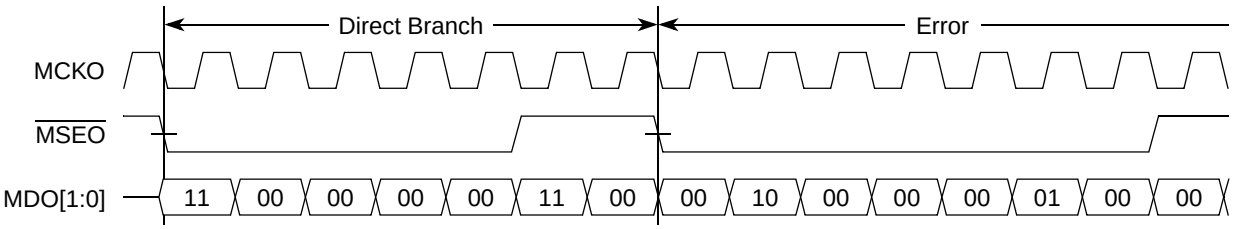
Figure 36-72. Program Trace (MDO = 12)—Indirect Branch Message (Traditional)



TCODE = 28
Source Processor = 0b0000
Number of Sequential Instructions = 0
Relative Address = 0xA5
Branch History = 0b1010_0101 (with Stop)

Note: This is representative only. The PXN20 supports only Full-Port Mode with 12 MDO pins.

Figure 36-73. Program Trace (MDO = 2)—Indirect Branch Message (History)

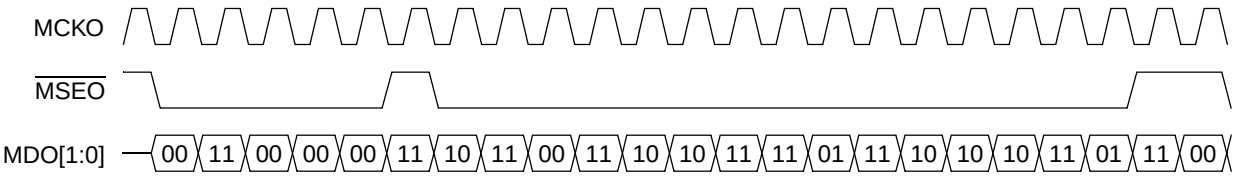


DBM:
TCODE = 3
Source Processor = 0b0000
Number of Sequential Instructions = 3

Error:
TCODE = 8
Source Processor = 0b0000
Error Code = 1 (Queue Overrun – BTM Only)

Note: This is representative only. The PXN20 supports only Full-Port Mode with 12 MDO pins.

Figure 36-74. Program Trace—Direct Branch (Traditional) and Error Messages



TCODE = 12
Source Processor = 0b0000
Number of Sequential Instructions = 3
Full Target Address = 0xDEAD_FACE

Note: This is representative only. The PXN20 supports only Full-Port Mode with 12 MDO pins.

Figure 36-75. Program Trace—Indirect Branch with Sync. Message

36.7.9.4 Watchpoint Support

This section details the watchpoint features of the Nexus2+ module.

36.7.9.4.1 Overview

The Nexus2+ module provides watchpoint messaging via the auxiliary pins, as defined by the IEEE-ISTO 5001-2003 standard.

Nexus2+ is not compliant with Class4 breakpoint/watchpoint requirements defined in the standard. The breakpoint/watchpoint control register is not implemented.

36.7.9.4.2 Watchpoint Messaging

Enabling watchpoint messaging is done by setting the watchpoint enable bit in the DC1 register. Setting the individual watchpoint sources is supported through the e200z0 Nexus1 module. The e200z0 Nexus1 module is capable of setting multiple address and/or data watch points. Please refer to the *e200z0Core Reference Manual* for more information on watchpoint initialization.

When these watch points occur, a watchpoint event signal from the Nexus1 module causes a message to be sent to the queue to be messaged out. This message includes the watchpoint number indicating which watchpoint caused the message.

The occurrence of any of the e200z0 defined watch points can be programmed to assert the event out $\overline{\text{EVTO}}$ pin for one (1) period of the output clock (MCKO).

Watchpoint information is messaged out in the following format

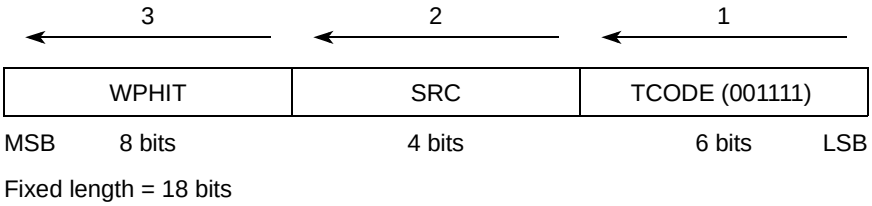


Figure 36-76. Watchpoint Message Format.

Table 36-61. Watchpoint Source Encoding

Watchpoint Source (8 bits)	Watchpoint Description
0b0000_0001	e200z0 Watchpoint #0 (IAC1 from Nexus1)
0b0000_0010	e200z0 Watchpoint #1 (IAC2 from Nexus1)
0b0000_0100	e200z0 Watchpoint #2 (IAC3 from Nexus1)
0b0000_1000	e200z0 Watchpoint #3 (IAC4 from Nexus1)
0b0001_0000	e200z0 Watchpoint #4 (DAC1 from Nexus1)
0b0010_0000	e200z0 Watchpoint #5 (DAC2 from Nexus1)

36.7.9.4.3 Watchpoint Error Message

An error message occurs when a new message cannot be queued due to the message queue being full. The FIFO discards messages until it has completely emptied the queue. Once emptied, an error message is queued. The error encoding indicates the type of messages queued to the FIFO while it was emptying.

If only a watchpoint message attempts to enter the queue while it is being emptied, the error message incorporates the watchpoint only error encoding (00110). If an OTM and/or program trace and/or data trace message also attempts to enter the queue while it is being emptied, the error message incorporates error encoding (01000).

NOTE

Set the OVC bits within the DC1 register to delay the CPU to alleviate (but not eliminate) potential overrun situations.

Error information is messaged out in the following format (see Table 36-47)

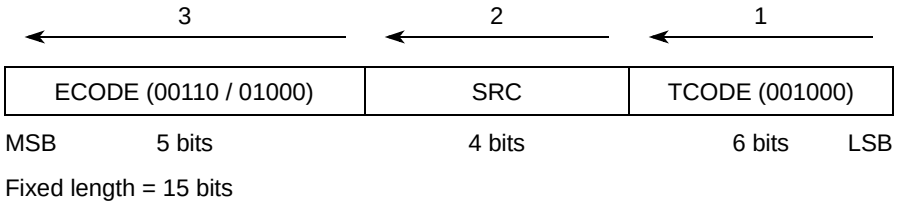
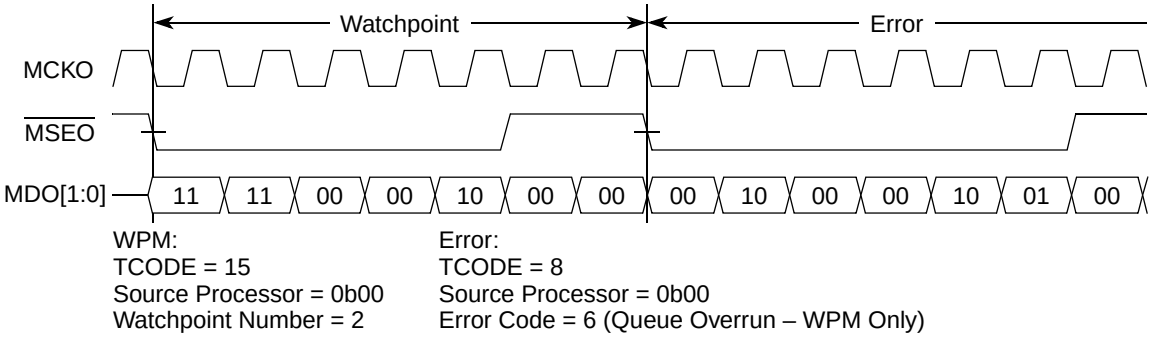


Figure 36-77. Error Message Format

36.7.9.4.4 Watchpoint Timing Diagram (2 MDO/1 MSEO Configuration)



Note: This is representative only. The PXN20 supports only Full-Port Mode with 12 MDO pins.

Figure 36-78. Watchpoint Message and Watchpoint Error Message

36.7.9.5 Nexus2+ Read/Write Access to Memory-Mapped Resources

The read/write access feature allows access to memory-mapped resources via the JTAG/OnCE port. The read/write mechanism supports single as well as block reads and writes to e200z0 system bus resources.

The Nexus2+ module is capable of accessing resources on the e200z0 system bus, with multiple configurable priority levels. Memory-mapped registers and other non-cached memory can be accessed via the standard memory map settings.

All accesses are setup and initiated by the read/write access control/status register (RWCS), as well as the read/write access address (RWA) and read/write access data registers (RWD).

Using the read/write access registers (RWCS/RWA/RWD), memory-mapped e200z0 system bus resources can be accessed through Nexus2+. The following subsections describe the steps required to access memory-mapped resources.

NOTE

Read/write access can only access memory mapped resources when system reset is de-asserted.

Misaligned accesses are NOT supported in the e200z0 Nexus2+ module.

36.7.9.5.1 Single Write Access

1. Initialize the read/write access address register (RWA) through the access method outlined in Section 36.7.8, Nexus2+ Register Access via JTAG / OnCE using the Nexus register index of 0x9 (see Table 36-48). Configure as follows:
   – Write Address → 0x*nnnn_nnnn* (write address)
2. Initialize the read/write access control/status register (RWCS) through the access method outlined in Section 36.7.8, Nexus2+ Register Access via JTAG / OnCE, using the Nexus Register Index of 0x7(see Table 36-48). Configure the bits as follows:
   – Access Control RWCS[AC] → 0b1 (to indicate start access)
   – Map Select RWCS[MAP] → 0b000 (primary memory map)
   – Access Priority RWCS[PR] → 0b00 (lowest priority)
   – Read/Write RWCS[RW] → 0b1 (write access)
   – Word Size RWCS[SZ] → 0b0xx (32-bit, 16-bit, 8-bit)
   – Access Count RWCS[CNT] → 0x0000 or 0x0001 (single access)

NOTE

Access count RWCS[CNT] of 0x0000 or 0x0001 performs a single access.

3. Initialize the read/write access data register (RWD) through the access method outlined in Section 36.7.8, Nexus2+ Register Access via JTAG / OnCE, using the Nexus register index of 0xA (see Table 36-48). Configure as follows:
   – Write Data → 0x*nnnn_nnnn* (write data)
4. The Nexus2+ module then arbitrates for the system bus and transfer the data value from the data buffer RWD register to the memory mapped address in the read/write access address register (RWA). When the access has completed without error (ERR = 0), Nexus2+ clears the DV bit in the RWCS register. This indicates that the device is ready for the next access.

NOTE

The DV and ERR bits within the RWCS provide read/write access status to the external development tool.

36.7.9.5.2 Block Write Access

1. For a non-burst block write access, follow Steps 1, 2, and 3 outlined in Section 36.7.9.5.1, Single Write Access to initialize the registers, but using a value greater than one (0x1) for the RWCS[CNT] field.
2. The Nexus2+ module then arbitrates for the system bus and transfer the first data value from the RWD register to the memory mapped address in the read/write access address register (RWA). When the transfer has completed without error (ERR = 0), the address from the RWA register is incremented to the next word size (specified in the SZ field) and the number from the CNT field is decremented.

3. Repeat step 3 in Section 36.7.9.5.1, Single Write Access until the internal CNT value is zero (0). When this occurs, the DV bit within the RWCS is cleared to indicate the end of the block write access.

36.7.9.5.3 Single Read Access

1. Initialize the read/write access address register (RWA) through the access method outlined in Section 36.7.8, Nexus2+ Register Access via JTAG / OnCE, using the Nexus register index of 0x9 (see Table 36-48). Configure as follows:
   - Read Address → 0x*nnnn_nnnn* (read address)
2. Initialize the read/write access control/status register (RWCS) through the access method outlined in Section 36.7.8, Nexus2+ Register Access via JTAG / OnCE, using the Nexus register index of 0x7 (see Table 36-48). Configure the bits as follows:
   - Access Control RWCS[AC] → 0b1 (to indicate start access)
   - Map Select RWCS[MAP] → 0b000 (primary memory map)
   - Access Priority RWCS[PR] → 0b00 (lowest priority)
   - Read/Write RWCS[RW] → 0b0 (read access)
   - Word Size RWCS[SZ] → 0b0xx (32-bit, 16-bit, 8-bit)
   - Access Count RWCS[CNT] → 0x0000 or 0x0001 (single access)

NOTE

Access Count (CNT) of 0x0000 or 0x0001 performs a single access.

3. The Nexus2+ module then arbitrates for the system bus and the read data is transferred from the system bus to the RWD register. When the transfer is completed without error (ERR = 0), Nexus sets the DV bit in the RWCS register. This indicates that the device is ready for the next access.
4. The data can then be read from the read/write access data register (RWD) through the access method outlined in Section 36.7.8, Nexus2+ Register Access via JTAG / OnCE, using the Nexus register index of 0xA (see Table 36-48).

NOTE

The DV and ERR bits within the RWCS provide Read/Write Access status to the external development tool.

36.7.9.5.4 Block Read Access

1. For a non-burst block read access, follow Steps 1 and 2 outlined in Section 36.7.9.5.3, Single Read Access to initialize the registers, but using a value greater than one (0x1) for the CNT field in the RWCS register.

   The Nexus2+ module then arbitrates for the system bus and the read data is transferred from the system bus to the RWD register. When the transfer has completed without error (ERR = 0), the address from the RWA register is incremented to the next word size (specified in the SZ field) and the number from the CNT field is decremented.

2. The data can then be read from the read/write access data register (RWD) through the access method outlined in Section 36.7.8, Nexus2+ Register Access via JTAG / OnCE, using the Nexus register index of 0xA (see Table 36-48).
3. Repeat steps 3 and 4 in Section 36.7.9.5.3, Single Read Access until the CNT value is zero (0). When this occurs, the DV bit within the RWCS is set to indicate the end of the block read access.

36.7.9.5.5 Error Handling

The Nexus2+ module handles various error conditions as follows:

System Bus Read/Write Error

All address and data errors that occur on read/write accesses to the e200z0 system bus return a transfer error. If this occurs:

1. The access is terminated without re-trying (AC bit is cleared).
2. The ERR bit in the RWCS register is set.
3. The error message is sent (TCODE = 8) indicating read/write error.

Access Termination

The following cases are defined for sequences of the read/write protocol that differ from those described in the above sections:

1. If the AC bit in the RWCS register is set to start read/write accesses and invalid values are loaded into the RWD and/or RWA, then a system bus access error may occur. This is handled as described above.
2. If a block access is in progress (all cycles not completed), and the RWCS register is written, then the original block access is terminated at the boundary of the nearest completed access.
   a) If the RWCS is written with the AC bit set, the next read/write access begins and the RWD can be written to/ read from.
   b) If the RWCS is written with the AC bit cleared, the read/write access is terminated at the nearest completed access. This method can be used to break (early terminate) block accesses.

36.7.9.5.6 Read/Write Access Error Message

The read/write access error message is sent out when an system bus access error (read or write) has occurred.

Error information is messaged out in the following format:

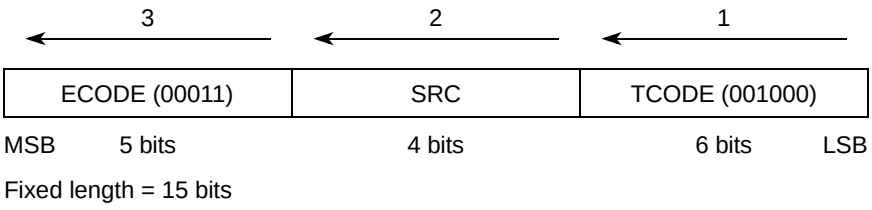


Figure 36-79. Error Message Format

36.7.9.6 Examples

The following are examples of program trace and data trace messages.

Table 36-62 illustrates an example indirect branch message with an 12MDO / 2 MSEO configuration.

Note that T0 and S0 are the least significant bits where:

- T*x* = TCODE number (fixed)
- S*x* = Source processor (fixed)
- I*x* = Number of instructions (variable)
- A*x* = Unique portion of the address (variable)

Table 36-62. Indirect Branch Message Example (12 MDO / 2 MSEO)

Clock	MDO[11:0]												MSEO[1:0]		State
	11	10	9	8	7	6	5	4	3	2	1	0			
0	X	X	X	X	X	X	X	X	X	X	X	X	1	1	Idle (or end of last message)
1	I1	I0	S3	S2	S1	S0	T5	T4	T3	T2	T1	T0	0	0	Start Message
2	0	0	0	0	0	0	0	0	I5	I4	I3	I2	0	1	End Packet
3	0	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	1	1	End Packet/End Message
4	X	X	S3	S2	S1	S0	T5	T4	T3	T2	T1	T0	0	0	Start of Next Message

Table 36-63 illustrates an example of direct branch message with 12 MDO / 2 MSEO.

Note that T0 and I0 are the least significant bits where:

- T*x* = TCODE number (fixed)
- S*x* = Source processor (fixed)
- I*x* = Number of instructions (variable)

Table 36-63. Direct Branch Message Example (12 MDO / 2 MSEO)

Clock	MDO[11:0]												MSEO[1:0]		State
	11	10	9	8	7	6	5	4	3	2	1	0			
0	X	X	X	X	X	X	X	X	X	X	X	X	1	1	Idle (or end of last message)
1	I1	I0	S3	S2	S1	S0	T5	T4	T3	T2	T1	T0	0	0	Start Message
2	0	0	0	0	0	0	0	0	0	0	I3	I2	1	1	End Packet/End Message
3	X	X	X	X	S1	S0	T5	T4	T3	T2	T1	T0	0	0	Start of Next Message

Table 36-64 an example data write message with 12 MDO / 2 MSEO configuration.

Note that T0, A0, D0 are the least significant bits where:

- T*x* = TCODE number (fixed)
- S*x* = Source processor (fixed)
- Z*x* = Data size (fixed)
- A*x* = Unique portion of the address (variable)
- D*x* = Write data (variable 8-bit, 16-bit, or 32-bit)

Table 36-64. Direct Write Message Example (12 MDO / 2 MSEO)

Clock	MDO[11:0]												MSEO[1:0]		State
	11	10	9	8	7	6	5	4	3	2	1	0			
0	X	X	X	X	X	X	X	X	X	X	X	X	1	1	Idle (or end of last message)
1	Z1	Z0	S3	S2	S1	S0	T5	T4	T3	T2	T1	T0	0	0	Start Message
2	0	0	0	0	0	0	0	A3	A2	A1	A0	Z2	0	1	End Packet
3	X	X	X	X	D7	D6	D5	D4	D3	D2	D1	D0	1	1	End Packet/End Message

36.7.9.7 IEEE 1149.1 (JTAG) RD/WR Sequences

This section contains example JTAG/OnCE sequences used to access resources.

36.7.9.7.1 JTAG Sequence for Accessing Internal Nexus Registers

Table 36-65. Accessing Internal Nexus2+ Registers via JTAG/OnCE

Step #	TMS Pin	Description
1	1	IDLE → SELECT-DR_SCAN
2	0	SELECT-DR_SCAN → CAPTURE-DR (Nexus command register value loaded in shifter)
3	0	CAPTURE-DR → SHIFT-DR
4	0	(7) TCK clocks issued to shift in direction (rd/wr) bit and first 6 bits of Nexus reg. addr.
5	1	SHIFT-DR → EXIT1-DR (7th bit of Nexus reg. shifted in)
6	1	EXIT1-DR → UPDATE-DR (Nexus shifter is transferred to Nexus command register)
7	1	UPDATE-DR → SELECT-DR_SCAN
8	0	SELECT-DR_SCAN → CAPTURE-DR (Register value is transferred to Nexus shifter)
9	0	CAPTURE-DR → SHIFT-DR
10	0	(31) TCK clocks issued to transfer register value to TDO pin while shifting in TDI value
11	1	SHIFT-DR → EXIT1-DR (MSB of value is shifted in/out of shifter)
12	1	EXIT1-DR → UPDATE -DR (if access is write, shifter is transferred to register)
13	0	UPDATE-DR → RUN-TEST/IDLE (transfer complete - Nexus controller to reg. select state)

36.7.9.7.2 JTAG Sequence for Read Access of Memory-Mapped Resources

Table 36-66. Accessing Memory-Mapped Resources (Reads)

Step #	TCLK clocks	Description
1	13	Nexus Command = write to read/write access address register (RWA)
2	37	Write RWA (initialize starting read address—data input on TDI)
3	13	Nexus Command = write to read/write control/status register (RWCS)
4	37	Write RWCS (initialize read access mode and CNT value—data input on TDI)
5	13	Nexus Command = read read/write access data register (RWD)
6	37	Read RWD (data output on TDO)
7	—	If CNT > 0, go back to Step #5

36.7.9.7.3 JTAG Sequence for Write Access of Memory-Mapped Resources

Table 36-67. Accessing Memory-Mapped Resources (Writes)

Step #	TCLK clocks	Description
1	13	Nexus Command = write to read/write access control/status register (RWCS)
2	37	Write RWCS (initialize write access mode and CNT value—data input on TDI)
3	13	Nexus Command = write to read/write address register (RWA)
4	37	Write RWA (initialize starting write address—data input on TDI)
5	13	Nexus Command = read read/write access data register (RWD)
6	37	Write RWD (data output on TDO)
7	—	If CNT > 0, go back to Step #5

36.8 Debug Implementation

This section describes the practical implementation of the debug port, its management, and the multiplexing strategy of its pads.

Standard JTAG (compliant to IEEE 1149.1) support is provided on all versions of the PXN20, and is described in Chapter 35, IEEE 1149.1 Test Access Port Controller (JTAGC). The Nexus Debug Interface (NDI), which is composed of the Nexus2+ (N2+) and Nexus3 (N3) clients running on the two cores, is available on the 256-pin package. This provides real-time development capabilities in compliance with the IEEE-ISTO 5001-2003 standard.

36.9 Debug Capabilities

The debug classifications according to the Nexus consortium are shown in Figure 36-80.

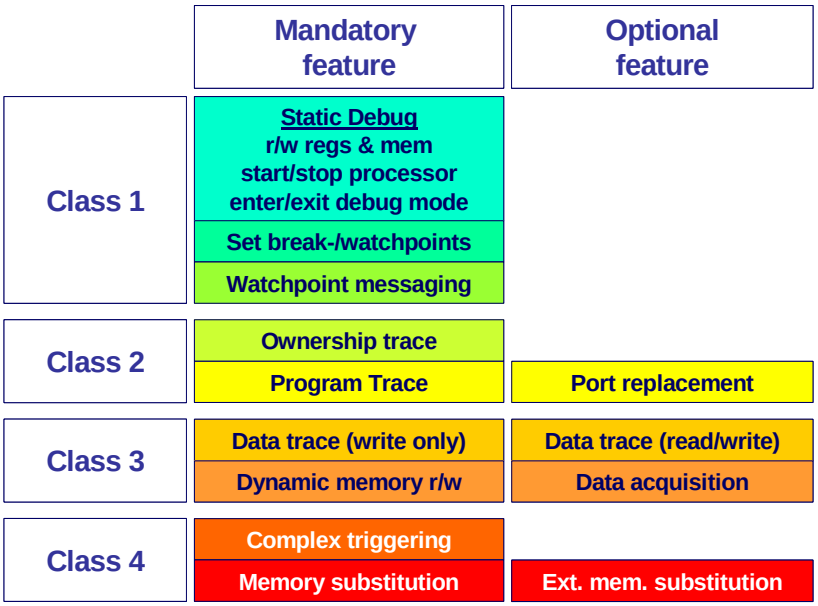


Figure 36-80. Nexus Debug Classifications

The PXN20 supports the Class 2+ and Class 3 debug features listed in Table 36-68

Table 36-68. Debug Requirements

Feature	Class 1	Class 2	Class 2+	Class 3	Class 3+	Class 4
Static debug	X	X	X	X	X	X
Set breakpoints/watchpoints	X	X	X	X	X	X
Watchpoint messaging	X	X	X	X	X	X
Ownership trace	—	X	X	X	X	X
Program trace (BTM)	—	X	X	X	X	X
Port replacement	—	optional	—	optional	optional	optional
Data trace (write only)	—	—	—	X	X	X
Dynamic memory read/write	—	—	X	X	X	X
Data trace (read/write)	—	—	—	optional	optional	optional
Data acquisition	—	—	—	optional	optional	optional
Memory substitution	—	—	—	—	—	X
Complex triggering	—	—	X	—	X	X
External memory substitution	—	—	—	—	—	optional

These debug capabilities are available at room temperature (25° C) and high temperature (125° C) when the PXN20 is at maximum speed.

The TAP controller operates at both 3.3 V and 5 V, according to the device power supply, but the Nexus output port operates only at 3.3 V maximum and is provided only on the 256MAPBGA emulation package.

Based on the following assumptions:

- Average message length: 20 bits
- 1 jump every 10 instructions
- MCKO capable of supporting 1/2 the system clock frequency (64 MHz maximum)
- MDO pins operate at half of the system clock speed

It is required to have 12 dedicated medium MDO pins. The N3/N2+ auxiliary port is bonded out only in the 256-pin MAPBGA package.

Boundary scan test is supported.

36.10 Debug Port

The debug port is composed of a total number of 22 pads as described in the following Table 36-69. The five JTAG pads are available on every device of the family and are not multiplexed.

The $\overline{\text{EVTO}}$ and $\overline{\text{EVTI}}$ functions are bonded out on dedicated 3.3 V pads on the 256MAPBGA. They are provided on the 208MAPBGA on 5 V pads multiplexed with GPIO. The remaining NDI pins are available as dedicated 3.3 V pads only on the 256MAPBGA package.

Table 36-69. Pin/Pad Multiplexing

Pin	Debug Port	Multiplexed?	Available on 208 MPABGA	Available on 256 MAPBGA
TDI	JTAG	Dedicated	Dedicated	Dedicated
TDO	JTAG	Dedicated	Dedicated	Dedicated
TMS/	JTAG	Dedicated	Dedicated	Dedicated
TCK	JTAG	Dedicated	Dedicated	Dedicated
JCOMP	JTAG	Dedicated	Dedicated	Dedicated
MCKO	N2+/3 Aux port	Dedicated	NO	Dedicated
MDO[0]	N2+/3 Aux port	Dedicated	NO	Dedicated
MDO[1]	N2+/3 Aux port	Dedicated	NO	Dedicated
MDO[2]	N2+/3 Aux port	Dedicated	NO	Dedicated
MDO[3]	N2+/3 Aux port	Dedicated	NO	Dedicated
MDO[4]	N2+/3 Aux port	Dedicated	NO	Dedicated
MDO[5]	N2+/3 Aux port	Dedicated	NO	Dedicated
MDO[6]	N2+/3 Aux port	Dedicated	NO	Dedicated
MDO[7]	N2+/3 Aux port	Dedicated	NO	Dedicated
MDO[8]	N2+/3 Aux port	Dedicated	NO	Dedicated

Table 36-69. Pin/Pad Multiplexing

Pin	Debug Port	Multiplexed?	Available on 208 MPABGA	Available on 256 MAPBGA
MDO[9]	N2+/3 Aux port	Dedicated	NO	Dedicated
MDO[10]	N2+/3 Aux port	Dedicated	NO	Dedicated
MDO[11]	N2+/3 Aux port	Dedicated	NO	Dedicated
$\overline{\text{MSEO}}$[0]	N2+/3 Aux port	Dedicated	NO	Dedicated
$\overline{\text{MSEO}}$[1]	N2+/3 Aux port	Dedicated	NO	Dedicated
$\overline{\text{EVTO}}$	N2+/3 Aux port	Dedicated (could MUX with I/O)	YES (MUX with I/O)	Dedicated
$\overline{\text{EVTI}}$	N2+/3Aux port	Dedicated (could MUX with I/O)	YES (MUX with I/O)	Dedicated

The reset and ready pins that are often present as extensions to the JTAG port are not implemented. Figure 36-81 shows the complete specification for the debug port in terms of pads.

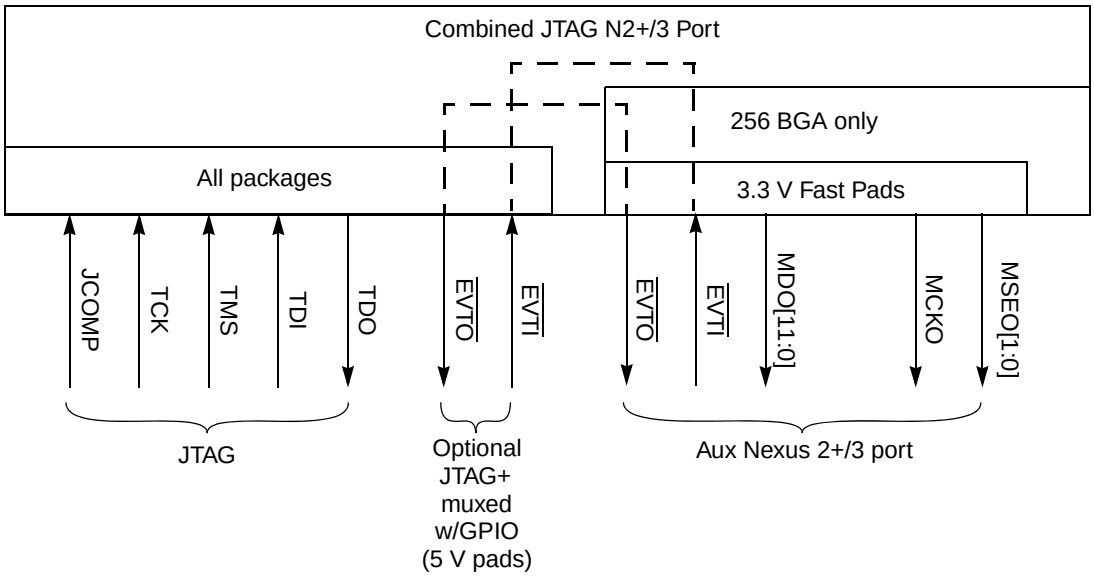


Figure 36-81. Debug Port Pads

36.10.1 Nexus2+/3 Auxiliary Port

The N2+/3 port provides real-time development class 2+ and class 3 capabilities in compliance with the IEEE-ISTO 5001-2003 standard. This development support is supplied without requiring external address and data pins for internal visibility.

The Nexus pads are ready to be used for debug purposes only after the activation of the Nexus controller. Such activation is consequent to a certain sequence given by the debugger.

By default, after power-on reset, N2+/3 circuitry (controller) and the dedicated pad are disabled to avoid power consumption. It can only be enabled via a certain sequence given by the debugger. As soon as the N2+/3 port is enabled even the N2+/3 pads are enabled.

36.11 Debug Methods

This section describes the debug methods for the available packages for the PXN20. The Nexus1 debug method applies to the 208MAPBGA, while the Nexus2+/3 applies to and is available only with the 256MAPBGA package.

36.11.1 208 MAPBGA Package Debug Method

The 208MAPBGA package provides only the Nexus class 1 debug method, based on IEEE1149.1 standard. Figure 36-82 shows an example.

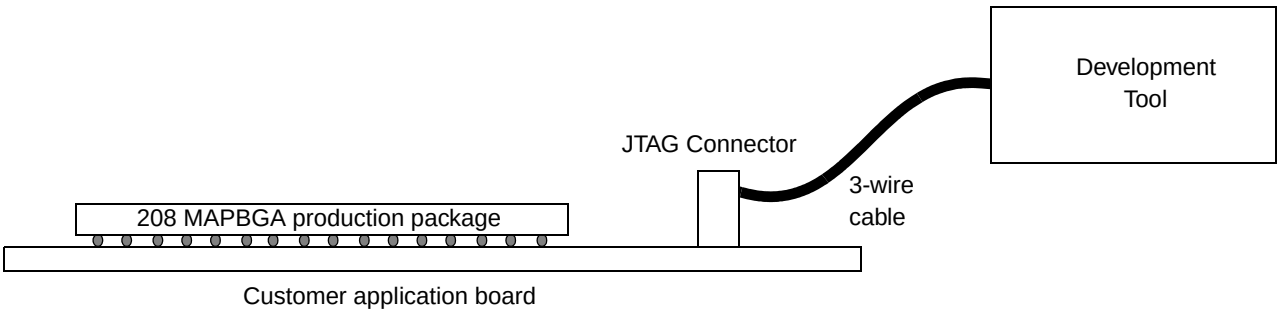


Figure 36-82. Nexus Class 1 Debugging Method Using JTAG Pins

36.11.2 256 MAPBGA Package Debug Method

The 256MAPBGA package provide allows the Nexus Class 2+/3 debug method as shown in Figure 36-83.

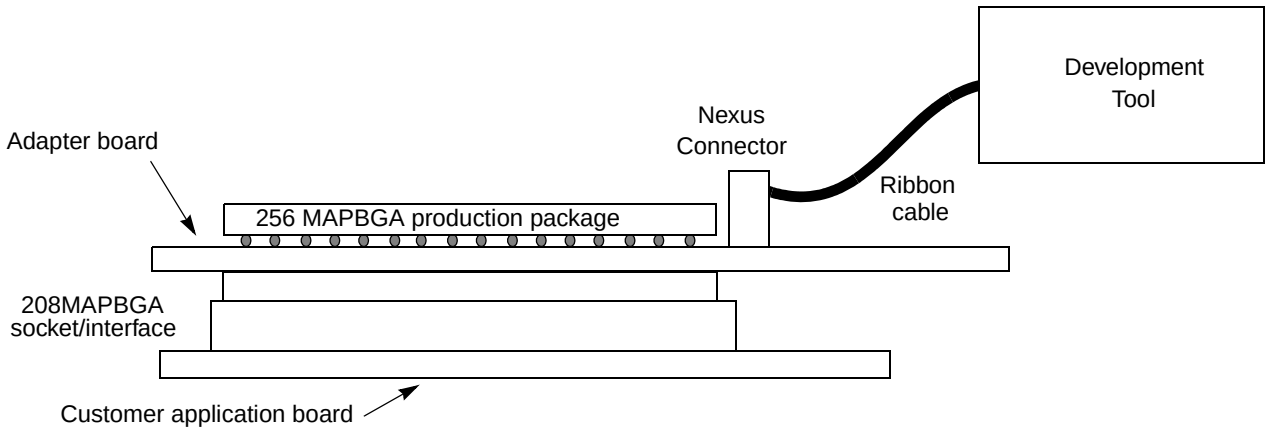


Figure 36-83. Nexus Class 2+/3 Debugging Method Using JTAG Pins

As 3.3 V fast pads are used for the N2+/3 port, an adapter board could provide buffers between the N2+/3 pads and the connector.

Appendix A
Memory Map

A.1 Introduction

This appendix lists the memory-mapped registers and addresses for the PXN20 device.

A.2 PXN20 Register Map

Table A-1. Module Base Addresses

Module Name	Base Address	Page
Program/Data Flash	0x0000_3FFF	Page A-3
Flash Emulation Mapping	0x0100_0000	Page A-3
SRAM	0x4000_0000	Page A-3
MLB_DIM Configuration	0xC3F8_4000	Page A-8
I^2C_C	0xC3F8_8000	Page A-11
I^2C_D	0xC3F8_C000	Page A-11
DSPI_C	0xC3F9_0000	Page A-12
DSPI_D	0xC3F9_4000	Page A-13
eSCI_J	0xC3FA_0000	Page A-14
eSCI_K	0xC3FA_4000	Page A-14
eSCI_L	0xC3FA_8000	Page A-15
eSCI_M	0xC3FA_C000	Page A-16
FlexRay	0xC3FD_C000	Page A-16
AXBS	0xFFF0_4000	Page A-39
Sema4	0xFFF1_0000	Page A-40
MPU	0xFFF1_4000	Page A-41
SWT	0xFFF3_8000	Page A-42
STM	0xFFF3_C000	Page A-43
ECSM	0xFFF4_0000	Page A-44
eDMA	0xFFF4_4000	Page A-44
INTC	0xFFF4_8000	Page A-47
FEC	0xFFF4_C000	Page A-50
ADC_A	0xFFF8_0000	Page A-51

Table A-1. Module Base Addresses (continued)

Module Name	Base Address	Page
I^2C_A	0xFFF8_8000	Page A-56
I^2C_B	0xFFF8_C000	Page A-56
DSPI_A	0xFFF9_0000	Page A-56
DSPI_B	0xFFF9_4000	Page A-57
eSCI_A	0xFFFA_0000	Page A-58
eSCI_B	0xFFFA_4000	Page A-59
eSCI_C	0xFFFA_8000	Page A-59
eSCI_D	0xFFFA_C000	Page A-60
eSCI_E	0xFFFB_0000	Page A-60
eSCI_F	0xFFFB_4000	Page A-61
eSCI_G	0xFFFB_8000	Page A-61
eSCI_H	0xFFFB_C000	Page A-62
FlexCan_A	0xFFFC_0000	Page A-62
FlexCan_B	0xFFFC_4000	Page A-67
FlexCan_C	0xFFFC_8000	Page A-72
FlexCan_D	0xFFFC_C000	Page A-76
FlexCan_E	0xFFFD_0000	Page A-81
FlexCan_F	0xFFFD_4000	Page A-86
CTU_A	0xFFFD_8000	Page A-90
DMA Multiplexer	0xFFFD_C000	Page A-92
PIT	0xFFFE_0000	Page A-93
eMIOS_A	0xFFFE_4000	Page A-94
SIU	0xFFFE_8000	Page A-101
CRP	0xFFFE_C000	Page A-111
FMPLL	0xFFFF_0000	Page A-112
PFlash Configuration	0xFFFF_8000	Page A-112
BAM	0xFFFF_C000	Page A-113

Table A-2. PXN20 System Memory Map

Address		Size {KB}	Region Name	Comments	Unimple-mented on PXN21	Unimple-mented on PXN20
Start	**End**					
Flash (AXBS Port S0 and S1)						
0x0000_0000	0x0000_3FFF	16	Program/Data Flash or Test Row	Test Row is accessible if the TST0[TRE] bit in the Flash configuration block is set		
0x0000_4000	0x0000_7FFF	16	Program/Data Flash			
0x0000_8000	0x0000_BFFF	16	Program/Data Flash			
0x0000_C000	0x0000_FFFF	16	Program/Data Flash			
0x0001_0000	0x0001_3FFF	16	Program/Data Flash			
0x0001_4000	0x0001_7FFF	16	Program/Data Flash			
0x0001_8000	0x0001_BFFF	16	Program/Data Flash			
0x0001_C000	0x0001_FFFF	16	Program/Data Flash			
0x0002_0000	0x0002_FFFF	64	Program/Data Flash			
0x0003_0000	0x0003_FFFF	64	Program/Data Flash			
0x0004_0000	0x0005_FFFF	128	Program/Data Flash			
0x0006_0000	0x0007_FFFF	128	Program/Data Flash			
0x0008_0000	0x000B_FFFF	256	Program/Data Flash			
0x000C_0000	0x000F_FFFF	256	Program/Data Flash			
0x0010_0000	0x0013_FFFF	256	Program/Data Flash			
0x0014_0000	0x0017_FFFF	256	Program/Data Flash			
0x0018_0000	0x001B_FFFF	256	Program/Data Flash			
0x001C_0000	0x001F_FFFF	256	Program/Data Flash			
0x0020_0000	0x00FF_BFFF	14,320	**Reserved**			
0x00FF_C000	0x00FF_FFFF	16	Shadow Row			
0x0100_0000	0x1FFF_FFFF	507,904	Flash Emulation Mapping			
0x2000_0000	0x3FFF_FFFF	524,288	**Reserved**			
SRAM (AXBS Ports S2 and S3)						
0x4000_0000	0x4007_FFFF	512	SRAM (AXBS Port S2)	This 1Mbyte address space is mirrored 512 times in the address range 0x4000_0000 to 0x5FFF_FFFF		> 128 KB
0x4008_0000	0x4009_3FFF	80	SRAM (AXBS Port S3)			X
0x4009_4000	0x400F_FFFF	432	**Reserved**			

Table A-2. PXN20 System Memory Map (continued)

Address		Size {KB}	Region Name	Comments	Unimple-mented on PXN21	Unimple-mented on PXN20
Start	End					
0x4010_0000	0xBFFF_FFFF		**Reserved**			
Peripherals AIPS_A (AXBS Port S6)						
0xC000_0000	0xC3EF_FFFF	64,512	**Reserved**			
0xC3F0_0000	0xC3F0_3FFF	16	**Reserved**			
0xC3F0_4000	0xC3F7_FFFF	496	**Reserved**			
0xC3F8_0000	0xC3F8_3FFF	16	**Reserved**			
0xC3F8_4000	0xC3F8_7FFF	16	MLB_DIM Configuration			X
0xC3F8_8000	0xC3F8_BFFF	16	I^2C_C			
0xC3F8_C000	0xC3F8_FFFF	16	I^2C_D			
0xC3F9_0000	0xC3F9_3FFF	16	DSPI_C			
0xC3F9_4000	0xC3F9_7FFF	16	DSPI_D			
0xC3F9_8000	0xC3F9_BFFF	16	**Reserved**			
0xC3F9_C000	0xC3F9_FFFF	16	**Reserved**			
0xC3FA_0000	0xC3FA_3FFF	16	eSCI_J		X	
0xC3FA_4000	0xC3FA_7FFF	16	eSCI_K		X	
0xC3FA_8000	0xC3FA_BFFF	16	eSCI_L		X	
0xC3FA_C000	0xC3FA_FFFF	16	eSCI_M		X	
0xC3FB_0000	0xC3FB_3FFF	16	**Reserved**			
0xC3FB_4000	0xC3FB_7FFF	16	**Reserved**			
0xC3FB_8000	0xC3FB_BFFF	16	**Reserved**			
0xC3FB_C000	0xC3FB_FFFF	16	**Reserved**			
0xC3FC_0000	0xC3FC_3FFF	16	**Reserved**			
0xC3FC_4000	0xC3FC_7FFF	16	**Reserved**			
0xC3FC_8000	0xC3FC_BFFF	16	**Reserved**			
0xC3FC_C000	0xC3FC_FFFF	16	**Reserved**			
0xC3FD_0000	0xC3FD_3FFF	16	**Reserved**			
0xC3FD_4000	0xC3FD_7FFF	16	**Reserved**			
0xC3FD_8000	0xC3FD_BFFF	16	**Reserved**			
0xC3FD_C000	0xC3FD_FFFF	16	**FlexRay**			X
0xC3FE_0000	0xC3FE_3FFF	16	**Reserved**			
0xC3FE_4000	0xC3FE_7FFF	16	**Reserved**			

Table A-2. PXN20 System Memory Map (continued)

Address		Size {KB}	Region Name	Comments	Unimple-mented on PXN21	Unimple-mented on PXN20
Start	**End**					
0xC3FE_8000	0xC3FF_3FFF	48	**Reserved**			
0xC3FF_4000	0xC3FF_7FFF	16	**Reserved**			
0xC3FF_8000	0xDFFF_FFFF	458,752	**Reserved**			
Peripherals AIPS_B (AXBS Port S7)						
0xE000_0000	0xFFEF_FFFF	523,264	**Reserved**			
0xFFF0_0000	0xFFF0_3FFF	16	**Reserved**			
0xFFF0_4000	0xFFF0_7FFF	16	AXBS			
0xFFF0_8000	0xFFF0_FFFF	32	**Reserved**			
0xFFF1_0000	0xFFF1_3FFF	16	Sema4			
0xFFF1_4000	0xFFF1_7FFF	16	MPU		X	
0xFFF1_8000	0xFFF3_7FFF	128	**Reserved**			
0xFFF3_8000	0xFFF3_BFFF	16	SWT			
0xFFF3_C000	0xFFF3_FFFF	16	STM			
0xFFF4_0000	0xFFF4_3FFF	16	ECSM			
0xFFF4_4000	0xFFF4_7FFF	16	eDMA		Channels 16-31	
0xFFF4_8000	0xFFF4_BFFF	16	INTC			
0xFFF4_C000	0xFFF4_FFFF	16	FEC			X
0xFFF5_0000	0xFFF7_FFFF	192	**Reserved**			
0xFFF8_0000	0xFFF8_3FFF	16	ADC_A			
0xFFF8_4000	0xFFF8_7FFF	16	**Reserved**			
0xFFF8_8000	0xFFF8_BFFF	16	I^2C_A			
0xFFF8_C000	0xFFF8_FFFF	16	I^2C_B			
0xFFF9_0000	0xFFF9_3FFF	16	DSPI_A			
0xFFF9_4000	0xFFF9_7FFF	16	DSPI_B			
0xFFF9_8000	0xFFF9_BFFF	16	**Reserved**			
0xFFF9C000	0xFFF9_FFFF	16	**Reserved**			
0xFFFA_0000	0xFFFA_3FFF	16	eSCI_A			
0xFFFA_4000	0xFFFA_7FFF	16	eSCI_B			
0xFFFA_8000	0xFFFA_BFFF	16	eSCI_C			
0xFFFA_C000	0xFFFA_FFFF	16	eSCI_D			
0xFFFB_0000	0xFFFB_3FFF	16	eSCI_E			

Table A-2. PXN20 System Memory Map (continued)

Address		Size {KB}	Region Name	Comments	Unimple-mented on PXN21	Unimple-mented on PXN20
Start	End					
0xFFFB_4000	0xFFFB_7FFF	16	eSCI_F			
0xFFFB_8000	0xFFFB_BFFF	16	eSCI_G		X	
0xFFFB_C000	0xFFFB_FFFF	16	eSCI_H		X	
0xFFFC_0000	0xFFFC_3FFF	16	FlexCan_A			
0xFFFC_4000	0xFFFC_7FFF	16	FlexCan_B			
0xFFFC_8000	0xFFFC_BFFF	16	FlexCan_C			
0xFFFC_C000	0xFFFC_FFFF	16	FlexCan_D			
0xFFFD_0000	0xFFFD_3FFF	16	FlexCan_E			
0xFFFD_4000	0xFFFD_7FFF	16	FlexCan_F			X
0xFFFD_8000	0xFFFD_BFFF	16	CTU_A		X	
0xFFFD_C000	0xFFFD_FFFF	16	DMA Multiplexer			
0xFFFE_0000	0xFFFE_3FFF	16	PIT			
0xFFFE_4000	0xFFFE_7FFF	16	eMIOS_A		Channels 24-31	
0xFFFE_8000	0xFFFE_BFFF	16	SIU			
0xFFFE_C000	0xFFFE_FFFF	16	CRP			
0xFFFF_0000	0xFFFF_3FFF	16	PLL			
0xFFFF_4000	0xFFFF_7FFF	16	**Reserved**			
0xFFFF_8000	0xFFFF_BFFF	16	PFlash Configuration			
0xFFFF_C000	0xFFFF_FFFF	16	BAM (upper 8K)			

Table A-3. Flash Memory Map

Offset from FLASH_BASE (0x0000_0000)	Use	Block	Partition
0x0000_0000	Low-address space	L0	1
0x0000_4000		L1	
0x0000_8000		L2	
0x0000_C000		L3	
0x0001_0000		L4	2
0x0001_4000		L5	
0x0001_8000		L6	
0x0001_C000		L7	
0x0002_0000		L8	3
0x0003_0000		L9	
0x0004_0000	Mid-address space	M0	4
0x0006_0000		M1	
0x0008_0000	High-address space	H0	5
0x000A_0000		H1	
0x000C_0000		H2	6
0x000E_0000		H3	
0x0010_0000		H4	7
0x0012_0000		H5	
0x0014_0000		H6	8
0x0016_0000		H7	
0x0018_0000–0xF0_FFFF	Reserved		
0x00FF_8000–0x00FF_FDD7	General use	S	All[1]
0x00FF_FDD8	Serial passcode (0xFEED_FACE_CAFE_BEEF)		
0x00FF_FDE0	Censorship control word (0x55AA_55AA)		
0x00FF_FDE4	General use		
0x00FF_FDE8	LML reset configuration (0x0010_0000)		
0x00FF_FDEC	General use		
0x00FF_FDF0	HBL reset configuration (0x0FFF_FFFF)		
0x00FF_FDF4	General use		
0x00FF_FDF8	SLL reset configuration (0x000F_FFFF)		
0x00FF_FDFC– 0x00FF_FFFF	General use		

1 For read while write operations, the shadow row behaves as if it is in all partitions.

0x0020_0000	0x00FF_BFFF	14,320	**Reserved**			
0x00FF_C000	0x00FF_FFFF	16	Shadow Row			
0x0100_0000	0x1FFF_FFFF	507,904	Flash Emulation Mapping			

Table A-4. PXN20 Detailed Register Map

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
SRAM (AXBS Ports S2 and S3)				
0x4000_0000 – 0x4007_FFFF	SRAM (AXBS Port S2)	R/W		
Peripherals AIPS_A (AXBS Port S6) 0xC000_0000– 0xDFFF_FFFF				
0xC000_0000– 0xC3F7_FFFF	**Reserved**			
0xC3F8_4000	**Media Local Bus** Chapter 27, Media Local Bus (MLB)			
0x0000	DCCR—Device Control Configuration Register	R/W	0x0000_0000	27.3.2.1/27-8
0x0004	SSCR—System Status Configuration Register	R/W	0x0000_0000	27.3.2.2/27-10
0x0008	SDCR—System Data Configuration Register	R	0x0000_0000	27.3.2.3/27-11
0x000C	SMCR—System Mask Configuration Register	R/W	0x0000_0060	27.3.2.4/27-12
0x001C	VCCR—Version Control Configuration Register	R	0x0300_0202	27.3.2.5/27-13
0x0020	SBCR—Synchronous Base Address Configuration Register	R/W	0x0000_0000	27.3.2.6/27-14
0x0024	ABCR—Asynchronous Base Address Configuration Register	R/W	0x0000_0000	27.3.2.7/27-14
0x0028	CBCR—Control Base Address Configuration Register	R/W	0x0000_0000	27.3.2.8/27-15
0x002C	IBCR—Isochronous Base Address Configuration Register	R/W	0x0000_0000	27.3.2.9/27-16
0x0030	CICR—Channel Interrupt Configuration Register	R	0x0000_0000	27.3.2.10/27-16
0x0040	CECR0—Channel 0 Entry Configuration Register	R/W	0x0000_0000	27.3.2.11/27-17
0x0044	CSCR0—Channel 0 Status Configuration Register	R/W	0x8000_0000	27.3.2.12/27-19
0x0048	CCBCR0—Channel 0 Current Buffer Configuration Register	R/W	0x0000_0000	27.3.2.13/27-22
0x004C	CNBCR0—Channel 0 Next Buffer Configuration Register	R/W	0x0000_0000	27.3.2.14/27-23
0x0050	CECR1—Channel 1 Entry Configuration Register	R/W	0x0000_0000	27.3.2.11/27-17
0x0054	CSCR1—Channel 1 Status Configuration Register	R/W	0x8000_0000	27.3.2.12/27-19
0x0058	CCBCR1—Channel 1 Current Buffer Configuration Register	R/W	0x0000_0000	27.3.2.13/27-22
0x005C	CNBCR1—Channel 1 Next Buffer Configuration Register	R/W	0x0000_0000	27.3.2.14/27-23

Table A-4. PXN20 Detailed Register Map (continued)

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
0x0060	CECR2—Channel 2 Entry Configuration Register	R/W	0x0000_0000	27.3.2.11/27-17
0x0064	CSCR2—Channel 2 Status Configuration Register	R/W	0x8000_0000	27.3.2.12/27-19
0x0068	CCBCR2—Channel 2 Current Buffer Configuration Register	R/W	0x0000_0000	27.3.2.13/27-22
0x006C	CNBCR2—Channel 2 Next Buffer Configuration Register	R/W	0x0000_0000	27.3.2.14/27-23
0x0070	CECR3—Channel 3 Entry Configuration Register	R/W	0x0000_0000	27.3.2.11/27-17
0x0074	CSCR3—Channel 3 Status Configuration Register	R/W	0x8000_0000	27.3.2.12/27-19
0x0078	CCBCR3—Channel 3 Current Buffer Configuration Register	R/W	0x0000_0000	27.3.2.13/27-22
0x007C	CNBCR3—Channel 3 Next Buffer Configuration Register	R/W	0x0000_0000	27.3.2.14/27-23
0x0080	CECR4—Channel 4 Entry Configuration Register	R/W	0x0000_0000	27.3.2.11/27-17
0x0084	CSCR4—Channel 4 Status Configuration Register	R/W	0x8000_0000	27.3.2.12/27-19
0x0088	CCBCR4—Channel 4 Current Buffer Configuration Register	R/W	0x0000_0000	27.3.2.13/27-22
0x008C	CNBCR4—Channel 4 Next Buffer Configuration Register	R/W	0x0000_0000	27.3.2.14/27-23
0x0090	CECR5—Channel 5 Entry Configuration Register	R/W	0x0000_0000	27.3.2.11/27-17
0x0094	CSCR5—Channel 5 Status Configuration Register	R/W	0x8000_0000	27.3.2.12/27-19
0x0098	CCBCR5—Channel 5 Current Buffer Configuration Register	R/W	0x0000_0000	27.3.2.13/27-22
0x009C	CNBCR5—Channel 5 Next Buffer Configuration Register	R/W	0x0000_0000	27.3.2.14/27-23
0x00A0	CECR6—Channel 6 Entry Configuration Register	R/W	0x0000_0000	27.3.2.11/27-17
0x00A4	CSCR6—Channel 6 Status Configuration Register	R/W	0x8000_0000	27.3.2.12/27-19
0x00A8	CCBCR6—Channel 6 Current Buffer Configuration Register	R/W	0x0000_0000	27.3.2.13/27-22
0x00AC	CNBCR6—Channel 6 Next Buffer Configuration Register	R/W	0x0000_0000	27.3.2.14/27-23
0x00B0	CECR7—Channel 7 Entry Configuration Register	R/W	0x0000_0000	27.3.2.11/27-17
0x00B4	CSCR7—Channel 7 Status Configuration Register	R/W	0x8000_0000	27.3.2.12/27-19
0x00B8	CCBCR7—Channel 7 Current Buffer Configuration Register	R/W	0x0000_0000	27.3.2.13/27-22
0x00BC	CNBCR7—Channel 7 Next Buffer Configuration Register	R/W	0x0000_0000	27.3.2.14/27-23
0x00C0	CECR8—Channel 8 Entry Configuration Register	R/W	0x0000_0000	27.3.2.11/27-17
0x00C4	CSCR8—Channel 8 Status Configuration Register	R/W	0x8000_0000	27.3.2.12/27-19
0x00C8	CCBCR8—Channel 8 Current Buffer Configuration Register	R/W	0x0000_0000	27.3.2.13/27-22
0x00CC	CNBCR8—Channel 8 Next Buffer Configuration Register	R/W	0x0000_0000	27.3.2.14/27-23
0x00D0	CECR9—Channel 9 Entry Configuration Register	R/W	0x0000_0000	27.3.2.11/27-17
0x00D4	CSCR9—Channel 9 Status Configuration Register	R/W	0x8000_0000	27.3.2.12/27-19
0x00D8	CCBCR9—Channel 9 Current Buffer Configuration Register	R/W	0x0000_0000	27.3.2.13/27-22
0x00DC	CNBCR9—Channel 9 Next Buffer Configuration Register	R/W	0x0000_0000	27.3.2.14/27-23

Table A-4. PXN20 Detailed Register Map (continued)

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
0x00E0	CECR10—Channel 10 Entry Configuration Register	R/W	0x0000_0000	27.3.2.11/27-17
0x00E4	CSCR10—Channel 10 Status Configuration Register	R/W	0x8000_0000	27.3.2.12/27-19
0x00E8	CCBCR10—Channel 10 Current Buffer Configuration Register	R/W	0x0000_0000	27.3.2.13/27-22
0x00EC	CNBCR10—Channel 10 Next Buffer Configuration Register	R/W	0x0000_0000	27.3.2.14/27-23
0x00F0	CECR11—Channel 11 Entry Configuration Register	R/W	0x0000_0000	27.3.2.11/27-17
0x00F4	CSCR11—Channel 11 Status Configuration Register	R/W	0x8000_0000	27.3.2.12/27-19
0x00F8	CCBCR11—Channel 11 Current Buffer Configuration Register	R/W	0x0000_0000	27.3.2.13/27-22
0x00FC	CNBCR11—Channel 11 Next Buffer Configuration Register	R/W	0x0000_0000	27.3.2.14/27-23
0x0100	CECR12—Channel 12 Entry Configuration Register	R/W	0x0000_0000	27.3.2.11/27-17
0x0104	CSCR12—Channel 12 Status Configuration Register	R/W	0x8000_0000	27.3.2.12/27-19
0x0108	CCBCR12—Channel 12 Current Buffer Configuration Register	R/W	0x0000_0000	27.3.2.13/27-22
0x010C	CNBCR12—Channel 12 Next Buffer Configuration Register	R/W	0x0000_0000	27.3.2.14/27-23
0x0110	CECR13—Channel 13 Entry Configuration Register	R/W	0x0000_0000	27.3.2.11/27-17
0x0114	CSCR13—Channel 13 Status Configuration Register	R/W	0x8000_0000	27.3.2.12/27-19
0x0118	CCBCR13—Channel 13 Current Buffer Configuration Register	R/W	0x0000_0000	27.3.2.13/27-22
0x011C	CNBCR13—Channel 13 Next Buffer Configuration Register	R/W	0x0000_0000	27.3.2.14/27-23
0x0120	CECR14—Channel 14 Entry Configuration Register	R/W	0x0000_0000	27.3.2.11/27-17
0x0124	CSCR14—Channel 14 Status Configuration Register	R/W	0x8000_0000	27.3.2.12/27-19
0x0128	CCBCR14—Channel 14 Current Buffer Configuration Register	R/W	0x0000_0000	27.3.2.13/27-22
0x012C	CNBCR14—Channel 14 Next Buffer Configuration Register	R/W	0x0000_0000	27.3.2.14/27-23
0x0130	CECR15—Channel 15 Entry Configuration Register	R/W	0x0000_0000	27.3.2.11/27-17
0x0134	CSCR15—Channel 15 Status Configuration Register	R/W	0x8000_0000	27.3.2.12/27-19
0x0138	CCBCR15—Channel 15 Current Buffer Configuration Register	R/W	0x0000_0000	27.3.2.13/27-22
0x013C	CNBCR15—Channel 15 Next Buffer Configuration Register	R/W	0x0000_0000	27.3.2.14/27-23
0x0140–0x027F	Reserved			
0x0280	LCBCR0—Local Channel 0 Buffer Configuration Register	R/W	0x0803_E000	27.3.2.14/27-23
0x0284	LCBCR1—Local Channel 1 Buffer Configuration Register	R/W	0x0803_E020	27.3.2.15/27-24
0x0288	LCBCR2—Local Channel 2 Buffer Configuration Register	R/W	0x0803_E040	27.3.2.15/27-24

Table A-4. PXN20 Detailed Register Map (continued)

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
0x028C	LCBCR3—Local Channel 3 Buffer Configuration Register	R/W	0x0803_E060	27.3.2.15/27-24
0x0290	LCBCR4—Local Channel 4 Buffer Configuration Register	R/W	0x0803_E080	27.3.2.15/27-24
0x0294	LCBCR5—Local Channel 5 Buffer Configuration Register	R/W	0x0803_E0A0	27.3.2.15/27-24
0x0298	LCBCR6—Local Channel 6 Buffer Configuration Register	R/W	0x0803_E0C0	27.3.2.15/27-24
0x029C	LCBCR7—Local Channel 7 Buffer Configuration Register	R/W	0x0803_E0E0	27.3.2.15/27-24
0x02A0	LCBCR8—Local Channel 8 Buffer Configuration Register	R/W	0x0803_E100	27.3.2.15/27-24
0x02A4	LCBCR9—Local Channel 9 Buffer Configuration Register	R/W	0x0803_E120	27.3.2.15/27-24
0x02A8	LCBCR10—Local Channel 10 Buffer Configuration Register	R/W	0x0803_E140	27.3.2.15/27-24
0x02AC	LCBCR11—Local Channel 11 Buffer Configuration Register	R/W	0x0803_E160	27.3.2.15/27-24
0x02B0	LCBCR12—Local Channel 12 Buffer Configuration Register	R/W	0x0803_E180	27.3.2.15/27-24
0x02B4	LCBCR13—Local Channel 13 Buffer Configuration Register	R/W	0x0803_E1A0	27.3.2.15/27-24
0x02B8	LCBCR14—Local Channel 14 Buffer Configuration Register	R/W	0x0803_E1C0	27.3.2.15/27-24
0x02BC	LCBCR15—Local Channel 15 Buffer Configuration Register	R/W	0x0803_E1E0	27.3.2.15/27-24
0x02C0–0x3FFF	Reserved			
0xC3F8_8000	**I2C_C** **Chapter 32, Inter-Integrated Circuit Bus Controller Module (I2C)**			
0x0000	IBAD—I^{2}C bus address register	R/W	0x00	32.3.2.1/32-5
0x0001	IBFD—I^{2}C bus frequency divider register	R/W	0x00	32.3.2.2/32-5
0x0002	IBCR—I^{2}C bus control register	R/W	0x80	32.3.2.3/32-8
0x0003	IBSR—I^{2}C bus status register	R/W	0x80	32.3.2.4/32-9
0x0004	IBDR—I^{2}C bus data I/O register	R/W	0x00	32.3.2.5/32-10
0x0005	IBIC—I^{2}C bus interrupt configuration register	R/W	0x00	32.3.2.6/32-11
0x0006–0x3FFF	Reserved			
0xC3F8_C000	**I^{2}C_D** **Chapter 32, Inter-Integrated Circuit Bus Controller Module (I2C)**			
0x0000	IBAD—I^{2}C bus address register	R/W	0x00	32.3.2.1/32-5
0x0001	IBFD—I^{2}C bus frequency divider register	R/W	0x00	32.3.2.2/32-5
0x0002	IBCR—I^{2}C bus control register	R/W	0x80	32.3.2.3/32-8
0x0003	IBSR—I^{2}C bus status register	R/W	0x80	32.3.2.4/32-9
0x0004	IBDR—I^{2}C bus data I/O register	R/W	0x00	32.3.2.5/32-10
0x0005	IBIC—I^{2}C bus interrupt configuration register	R/W	0x00	32.3.2.6/32-11
0x0006–0x3FFF	Reserved			

Table A-4. PXN20 Detailed Register Map (continued)

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
0xC3F9_0000	**DSPI_C** **Chapter 30, Deserial – Serial Peripheral Interface (DSPI)**			
0x0000	DSPI_MCR—DSPI module configuration register	R/W	0x0000_4001	30.3.2.1/30-7
0x0004	Reserved			
0x0008	DSPI_TCR—DSPI transfer count register	R/W	0x0000_0000	30.3.2.2/30-9
0x000C	DSPI_CTAR0—DSPI clock and transfer attributes register 0	R/W	0x7800_0000	30.3.2.3/30-10
0x0010	DSPI_CTAR1—DSPI clock and transfer attributes register 1	R/W	0x7800_0000	30.3.2.3/30-10
0x0014	DSPI_CTAR2—DSPI clock and transfer attributes register 2	R/W	0x7800_0000	30.3.2.3/30-10
0x0018	DSPI_CTAR3—DSPI clock and transfer attributes register 3	R/W	0x7800_0000	30.3.2.3/30-10
0x001C	DSPI_CTAR4—DSPI clock and transfer attributes register 4	R/W	0x7800_0000	30.3.2.3/30-10
0x0020	DSPI_CTAR5—DSPI clock and transfer attributes register 5	R/W	0x7800_0000	30.3.2.3/30-10
0x0024	DSPI_CTAR6—DSPI clock and transfer attributes register 6	R/W	0x7800_0000	30.3.2.3/30-10
0x0028	DSPI_CTAR7—DSPI clock and transfer attributes register 7	R/W	0x7800_0000	30.3.2.3/30-10
0x002C	DSPI_SR—DSPI status register	R	0x0000_0000	30.3.2.4/30-16
0x0030	DSPI_RSER—DSPI DMA/interrupt request select and enable register	R/W	0x0000_0000	30.3.2.5/30-18
FIFO Registers				
0x0034	DSPI_PUSHR—DSPI push TX FIFO register	R/W	0x0000_0000	30.3.2.6/30-19
0x0038	DSPI_POPR—DSPI pop RX FIFO register	R	0x0000_0000	30.3.2.7/30-21
0x003C	DSPI_TXFR0—DSPI transmit FIFO register 0	R	0x0000_0000	30.3.2.8/30-21
0x0040	DSPI_TXFR1—DSPI transmit FIFO register 1	R	0x0000_0000	30.3.2.8/30-21
0x0044	DSPI_TXFR2—DSPI transmit FIFO register 2	R	0x0000_0000	30.3.2.8/30-21
0x0048	DSPI_TXFR3—DSPI transmit FIFO register 3	R	0x0000_0000	30.3.2.8/30-21
0x004C–0x0078	Reserved			
0x007C	DSPI_RXFR0—DSPI receive FIFO register 0	R	0x0000_0000	30.3.2.9/30-22
0x0080	DSPI_RXFR1—DSPI receive FIFO register 1	R	0x0000_0000	30.3.2.9/30-22
0x0084	DSPI_RXFR2—DSPI receive FIFO register 2	R	0x0000_0000	30.3.2.9/30-22
0x0088	DSPI_RXFR3—DSPI receive FIFO register 3	R	0x0000_0000	30.3.2.9/30-22
0x008C–0x00B8	Reserved			
DSI Registers				
0x00BC	DSPI_DSICR—DSPI DSI configuration register	R/W	0x0000_0000	30.3.2.10/30-23
0x00C0	DSPI_SDR—DSPI DSI serialization data register	R	0x0000_0000	30.3.2.11/30-24

Table A-4. PXN20 Detailed Register Map (continued)

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
0x00C4	DSPI_ASDR—DSPI DSI alternate serialization data register	R/W	0x0000_0000	30.3.2.12/30-25
0x00C8	DSPI_COMPR—DSPI DSI transmit comparison register	R	0x0000_0000	30.3.2.13/30-26
0x00CC	DSPI_DDR—DSPI DSI deserialization data register	R	0x0000_0000	30.3.2.14/30-26
0x00D0	DSPI_DSICR1—DSPI DSI TSB configuration register 1	R/W	0x0000_0000	30.3.2.15/30-27
0x00D4–0x3FFF	Reserved			
0xC3F9_4000	**DSPI_D** **Chapter 30, Deserial – Serial Peripheral Interface (DSPI)**			
0x0000	DSPI_MCR—DSPI module configuration register	R/W	0x0000_4001	30.3.2.1/30-7
0x0004	Reserved			
0x0008	DSPI_TCR—DSPI transfer count register	R/W	0x0000_0000	30.3.2.2/30-9
0x000C	DSPI_CTAR0—DSPI clock and transfer attributes register 0	R/W	0x7800_0000	30.3.2.3/30-10
0x0010	DSPI_CTAR1—DSPI clock and transfer attributes register 1	R/W	0x7800_0000	30.3.2.3/30-10
0x0014	DSPI_CTAR2—DSPI clock and transfer attributes register 2	R/W	0x7800_0000	30.3.2.3/30-10
0x0018	DSPI_CTAR3—DSPI clock and transfer attributes register 3	R/W	0x7800_0000	30.3.2.3/30-10
0x001C	DSPI_CTAR4—DSPI clock and transfer attributes register 4	R/W	0x7800_0000	30.3.2.3/30-10
0x0020	DSPI_CTAR5—DSPI clock and transfer attributes register 5	R/W	0x7800_0000	30.3.2.3/30-10
0x0024	DSPI_CTAR6—DSPI clock and transfer attributes register 6	R/W	0x7800_0000	30.3.2.3/30-10
0x0028	DSPI_CTAR7—DSPI clock and transfer attributes register 7	R/W	0x7800_0000	30.3.2.3/30-10
0x002C	DSPI_SR—DSPI status register	R	0x0200_0000	30.3.2.4/30-16
0x0030	DSPI_RSER—DSPI DMA/interrupt request select and enable register	R/W	0x0000_0000	30.3.2.5/30-18
FIFO Registers				
0x0034	DSPI_PUSHR—DSPI push TX FIFO register	R/W	0x0000_0000	30.3.2.6/30-19
0x0038	DSPI_POPR—DSPI pop RX FIFO register	R	0x0000_0000	30.3.2.7/30-21
0x003C	DSPI_TXFR0—DSPI transmit FIFO register 0	R	0x0000_0000	30.3.2.8/30-21
0x0040	DSPI_TXFR1—DSPI transmit FIFO register 1	R	0x0000_0000	30.3.2.8/30-21
0x0044	DSPI_TXFR2—DSPI transmit FIFO register 2	R	0x0000_0000	30.3.2.8/30-21
0x0048	DSPI_TXFR3—DSPI transmit FIFO register 3	R	0x0000_0000	30.3.2.8/30-21
0x004C–0x0078	Reserved			
0x007C	DSPI_RXFR0—DSPI receive FIFO register 0	R	0x0000_0000	30.3.2.9/30-22
0x0080	DSPI_RXFR1—DSPI receive FIFO register 1	R	0x0000_0000	30.3.2.9/30-22
0x0084	DSPI_RXFR2—DSPI receive FIFO register 2	R	0x0000_0000	30.3.2.9/30-22

Table A-4. PXN20 Detailed Register Map (continued)

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
0x0088	DSPI_RXFR3—DSPI receive FIFO register 3	R	0x0000_0000	30.3.2.9/30-22
0x008C–0x00B8	Reserved			
DSI Registers				
0x00BC	DSPI_DSICR—DSPI DSI configuration register	R/W	0x0000_0000	30.3.2.10/30-23
0x00C0	DSPI_SDR—DSPI DSI serialization data register	R	0x0000_0000	30.3.2.11/30-24
0x00C4	DSPI_ASDR—DSPI DSI alternate serialization data register	R/W	0x0000_0000	30.3.2.12/30-25
0x00C8	DSPI_COMPR—DSPI DSI transmit comparison register	R	0x0000_0000	30.3.2.13/30-26
0x00CC	DSPI_DDR—DSPI DSI deserialization data register	R	0x0000_0000	30.3.2.14/30-26
0x00D0	DSPI_DSICR1—DSPI DSI TSB configuration register 1	R/W	0x0000_0000	30.3.2.15/30-27
0x00D4–0x3FFF	Reserved			
0xC3F9_8000–0xC3F9_FFFF	**Reserved**			
0xC3FA_0000	**eSCI_J** **Chapter 31, Enhanced Serial Communication Interface (eSCI)**			
0x0000	eSCI_BRR—eSCI baud rate register	R/W	0x0004	31.3.2.1/31-6
0x0002	eSCI_CR1—eSCI control register 1	R/W	0x0000	31.3.2.2/31-6
0x0004	eSCI_CR2—eSCI control register 2	R/W	0x0200	31.3.2.3/31-8
0x0006	eSCI_SDR—eSCI data register	R/W	0x0000	31.3.2.4/31-10
0x0008	eSCI_IFSR1—eSCI interrupt flag and status register 1	R/W	0x8000	31.3.2.5/31-11
0x000A	eSCI_IFSR2—eSCI interrupt flag and status register 2	R/W	0x0000	31.3.2.6/31-12
0x000C	eSCI_LCR1—eSCI LIN control register 1	R/W	0x0000	31.3.2.7/31-13
0x000E	eSCI_LCR2—eSCI LIN control register 2	R/W	0x0000	31.3.2.8/31-15
0x00010	eSCI_LTR— eSCI LIN transmit register	R/W	0x0000	31.3.2.9/31-15
0x0012	Reserved			
0x0014	eSCI_LRR—eSCI LIN receive register	R/W	0x0000	31.3.2.10/31-17
0x0016	Reserved			
0x0018	eSCI_LPR—eSCI LIN CRC polynomial register	R/W	0xC599	31.3.2.11/31-18
0x001A	eSCI_CR3—eSCI control register 3	R/W	0x0000	31.3.2.12/31-18
0x001C–0x3FFF	Reserved			
0xC3FA_4000	**eSCI_K** **Chapter 31, Enhanced Serial Communication Interface (eSCI)**			
0x0000	eSCI_BRR—eSCI baud rate register	R/W	0x0004	31.3.2.1/31-6

Table A-4. PXN20 Detailed Register Map (continued)

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
0x0002	eSCI_CR1—eSCI control register 1	R/W	0x0000	31.3.2.2/31-6
0x0004	eSCI_CR2—eSCI control register 2	R/W	0x0200	31.3.2.3/31-8
0x0006	eSCI_SDR—eSCI data register	R/W	0x0000	31.3.2.4/31-10
0x0008	eSCI_IFSR1—eSCI interrupt flag and status register 1	R/W	0x8000	31.3.2.5/31-11
0x000A	eSCI_IFSR2—eSCI interrupt flag and status register 2	R/W	0x4000	31.3.2.6/31-12
0x000C	eSCI_LCR1—eSCI LIN control register 1	R/W	0x0000	31.3.2.7/31-13
0x000E	eSCI_LCR2—eSCI LIN control register 2	R/W	0x0000	31.3.2.8/31-15
0x00010	eSCI_LTR— eSCI LIN transmit register	R/W	0x0000	31.3.2.9/31-15
0x0012	Reserved			
0x0014	eSCI_LRR—eSCI LIN receive register	R/W	0x0000	31.3.2.10/31-17
0x0016	Reserved			
0x0018	eSCI_LPR—eSCI LIN CRC polynomial register	R/W	0xC599	31.3.2.11/31-18
0x001A	eSCI_CR3—eSCI control register 3	R/W	0x0000	31.3.2.12/31-18
0x001C–0x3FFF	Reserved			
0xC3FA_8000	**eSCI_L** **Chapter 31, Enhanced Serial Communication Interface (eSCI)**			
0x0000	eSCI_BRR—eSCI baud rate register	R/W	0x0004	31.3.2.1/31-6
0x0002	eSCI_CR1—eSCI control register 1	R/W	0x0000	31.3.2.2/31-6
0x0004	eSCI_CR2—eSCI control register 2	R/W	0x0200	31.3.2.3/31-8
0x0006	eSCI_SDR—eSCI data register	R/W	0x0000	31.3.2.4/31-10
0x0008	eSCI_IFSR1—eSCI interrupt flag and status register 1	R/W	0x0000	31.3.2.5/31-11
0x000A	eSCI_IFSR2—eSCI interrupt flag and status register 2	R/W	0x0000	31.3.2.6/31-12
0x000C	eSCI_LCR1—eSCI LIN control register 1	R/W	0x0000	31.3.2.7/31-13
0x000E	eSCI_LCR2—eSCI LIN control register 2	R/W	0x0000	31.3.2.8/31-15
0x00010	eSCI_LTR— eSCI LIN transmit register	R/W	0x0000	31.3.2.9/31-15
0x0012	Reserved			
0x0014	eSCI_LRR—eSCI LIN receive register	R/W	0x0000	31.3.2.10/31-17
0x0016	Reserved			
0x0018	eSCI_LPR—eSCI LIN CRC polynomial register	R/W	0xC599	31.3.2.11/31-18
0x001A	eSCI_CR3—eSCI control register 3	R/W	0x0000	31.3.2.12/31-18
0x001C–0x3FFF	Reserved			

Table A-4. PXN20 Detailed Register Map (continued)

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
0xC3FA_C000	**eSCI_M** **Chapter 31, Enhanced Serial Communication Interface (eSCI)**			
0x0000	eSCI_BRR—eSCI baud rate register	R/W	0x0004	31.3.2.1/31-6
0x0002	eSCI_CR1—eSCI control register 1	R/W	0x0000	31.3.2.2/31-6
0x0004	eSCI_CR2—eSCI control register 2	R/W	0x0200	31.3.2.3/31-8
0x0006	eSCI_SDR—eSCI data register	R/W	0x0000	31.3.2.4/31-10
0x0008	eSCI_IFSR1—eSCI interrupt flag and status register 1	R/W	0x0000	31.3.2.5/31-11
0x000A	eSCI_IFSR2—eSCI interrupt flag and status register 2	R/W	0x0000	31.3.2.6/31-12
0x000C	eSCI_LCR1—eSCI LIN control register 1	R/W	0x0000	31.3.2.7/31-13
0x000E	eSCI_LCR2—eSCI LIN control register 2	R/W	0x0000	31.3.2.8/31-15
0x00010	eSCI_LTR— eSCI LIN transmit register	R/W	0x0000	31.3.2.9/31-15
0x0012	Reserved			
0x0014	eSCI_LRR—eSCI LIN receive register	R/W	0x0000	31.3.2.10/31-17
0x0016	Reserved			
0x0018	eSCI_LPR—eSCI LIN CRC polynomial register	R/W	0xC599	31.3.2.11/31-18
0x001A	eSCI_CR3—eSCI control register 3	R/W	0x0000	31.3.2.12/31-18
0x001C–0x3FFF	Reserved			
0xC3FB_0000–0xC3FD_BFFF	**Reserved**			
0xC3FD_C000	**FlexRay** **Chapter 26, FlexRay Communication Controller (FlexRAY)**			
0x0000	MVR—Module version register	R	0x9E66	26.5.2.3/26-13
0x0002	MCR—Module configuration register	R/W	0x0000	26.5.2.4/26-14
0x0004	SYMBADHR—System memory base address high register	R/W	0x0000	26.5.2.5/26-15
0x0006	SYMBADLR—System memory base address low register	R/W	0x0000	26.5.2.5/26-15
0x0008	STBSCR—Strobe signal control register	R/W	0x0000	26.5.2.6/26-16
0x000A	Reserved			
0x000C	MBDSR—Message buffer data size register	R/W	0x0000	26.5.2.7/26-18
0x000E	MBSSUTR—Message buffer segment size and utilization register	R/W	0x7F7F	26.5.2.8/26-18
0x0010–0x0013	Reserved			
0x0014	POCR—Protocol operation control register	R/W	0x0000	26.5.2.9/26-19
0x0016	GIFER—Global interrupt flag and enable register	R/W	0x0000	26.5.2.10/26-20

Table A-4. PXN20 Detailed Register Map (continued)

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
0x0018	PIFR0—Protocol interrupt flag register 0	R/W	0x0000	26.5.2.11/26-23
0x001A	PIFR1—Protocol interrupt flag register 1	R/W	0x0000	26.5.2.12/26-25
0x001C	PIER0—Protocol interrupt enable register 0	R/W	0x0000	26.5.2.13/26-26
0x001E	PIER1—Protocol interrupt enable register 1	R/W	0x0000	26.5.2.14/26-27
0x0020	CHIERFR—CHI error flag register	R/W	0x0000	26.5.2.15/26-28
0x0022	MBIVEC—Message buffer interrupt vector register	R	0x0000	26.5.2.16/26-30
0x0024	CASERCR—Channel A status error counter register	R	0x0000	26.5.2.17/26-31
0x0026	CBSERCR—Channel B status error counter register	R	0x0000	26.5.2.18/26-31
0x0028	PSR0—Protocol status register 0	R	0x0000	26.5.2.19/26-32
0x002A	PSR1—Protocol status register 1	R	0x0000	26.5.2.20/26-33
0x002C	PSR2—Protocol status register 2	R	0x0000	26.5.2.21/26-34
0x002E	PSR3—Protocol status register 3	R/W	0x0000	26.5.2.22/26-36
0x0030	MTCTR—Macrotick counter register	R	0x0000	26.5.2.23/26-37
0x0032	CYCTR—Cycle counter register	R	0x0000	26.5.2.24/26-38
0x0034	SLTCTAR—Slot counter channel A register	R	0x0000	26.5.2.25/26-38
0x0036	SLTCTBR—Slot counter channel B register	R	0x0000	26.5.2.26/26-39
0x0038	RTCORVR—Rate correction value register	R	0x0000	26.5.2.27/26-39
0x003A	OFCORVR—Offset correction value register	R	0x0000	26.5.2.28/26-40
0x003C	CIFRR—Combined interrupt flag register	R	0x0000	26.5.2.29/26-40
0x003E	SYMATOR—System memory access time-out register	R/W	0x0004	26.5.2.30/26-41
0x0040	SFCNTR—Sync frame counter register	R	0x0000	26.5.2.31/26-42
0x0042	SFTOR—Sync frame table offset register	R/W	0x0000	26.5.2.32/26-42
0x0044	SFTCCSR—Sync frame table configuration, control, status register	R/W	0x0000	26.5.2.33/26-43
0x0046	SFIDRFR—Sync frame ID rejection filter register	R/W	0x0000	26.5.2.34/26-44
0x0048	SFIDAFVR—Sync frame ID acceptance filter value register	R/W	0x0000	26.5.2.35/26-45
0x004A	SFIDAFMR—Sync frame ID acceptance filter mask register	R/W	0x0000	26.5.2.36/26-45
0x004C	NMVR0—Network management vector register 0	R	0x0000	26.5.2.37/26-45
0x004E	NMVR1—Network management vector register 1	R	0x0000	26.5.2.37/26-45
0x0050	NMVR2—Network management vector register 2	R	0x0000	26.5.2.37/26-45
0x0052	NMVR3—Network management vector register 3	R	0x0000	26.5.2.37/26-45
0x0054	NMVR4—Network management vector register 4	R	0x0000	26.5.2.37/26-45

Table A-4. PXN20 Detailed Register Map (continued)

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
0x0056	NMVR5—Network management vector register 5	R	0x0000	26.5.2.37/26-45
0x0058	NMVLR—Network management vector length register	R/W	0x0000	26.5.2.38/26-46
0x005A	TICCR—Timer configuration and control register	R/W	0x0000	26.5.2.39/26-47
0x005C	TI1CYSR—Timer 1 cycle set register	R/W	0x0000	26.5.2.40/26-48
0x005E	TI1MTOR—Timer 1 macrotick offset register	R/W	0x0000	26.5.2.41/26-48
0x0060	TI2CR0—Timer 2 configuration register 0	R/W	0x0000	26.5.2.42/26-49
0x0062	TI2CR1—Timer 2 configuration register 1	R/W	0x0000	26.5.2.43/26-49
0x0064	SSSR—Slot status selection register	R/W	0x0000	26.5.2.44/26-50
0x0066	SSCCR—Slot status counter condition register	R/W	0x0000	26.5.2.45/26-51
0x0068	SSR0—Slot status register 0	R	0x0000	26.5.2.46/26-53
0x006A	SSR1—Slot status register 1	R	0x0000	26.5.2.46/26-53
0x006C	SSR2—Slot status register 2	R	0x0000	26.5.2.46/26-53
0x006E	SSR3—Slot status register 3	R	0x0000	26.5.2.46/26-53
0x0070	SSR4—Slot status register 4	R	0x0000	26.5.2.46/26-53
0x0072	SSR5—Slot status register 5	R	0x0000	26.5.2.46/26-53
0x0074	SSR6—Slot status register 6	R	0x0000	26.5.2.46/26-53
0x0076	SSR7—Slot status register 7	R	0x0000	26.5.2.46/26-53
0x0078	SSCR0—Slot status counter register 0	R	0x0000	26.5.2.47/26-54
0x007A	SSCR1—Slot status counter register 1	R	0x0000	26.5.2.47/26-54
0x007C	SSCR2—Slot status counter register 2	R	0x0000	26.5.2.47/26-54
0x007E	SSCR3—Slot status counter register 3	R	0x0000	26.5.2.47/26-54
0x0080	MTSACFR—MTS A configuration register	R/W	0x0000	26.5.2.48/26-55
0x0082	MTSBCFR—MTS B configuration register	R/W	0x0000	26.5.2.49/26-55
0x0084	RSBIR—Receive shadow buffer index register	R/W	0x0000	26.5.2.50/26-56
0x0086	RFWMSR—Receive FIFO watermark and selection register	R/W	0x0000	26.5.2.53/26-58
0x0088	RFSIR—Receive FIFO start index register	R/W	0x0000	26.5.2.54/26-58
0x008A	RFDSR—Receive FIFO depth and size register	R/W	0x0000	26.5.2.55/26-59
0x008C	RFARIR—Receive FIFO A read index register	R	0x0000	26.5.2.56/26-59
0x008E	RFBRIR—Receive FIFO B read index register	R	0x0000	26.5.2.57/26-60
0x0090	RFMIDAFVR—Receive FIFO message ID acceptance filter value register	R/W	0x0000	26.5.2.59/26-61
0x0092	RFMIAFMR—Receive FIFO message ID acceptance filter mask register	R/W	0x0000	26.5.2.60/26-61

Table A-4. PXN20 Detailed Register Map (continued)

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
0x0094	RFFIDRFVR—Receive FIFO frame ID rejection filter value register	R/W	0x0000	26.5.2.61/26-62
0x0096	RFFIDRFMR—Receive FIFO frame ID rejection filter mask register	R/W	0x0000	26.5.2.62/26-62
0x0098	RFRFCFR—Receive FIFO range filter configuration register	R/W	0x0000	26.5.2.63/26-62
0x009A	RFRFCTR—Receive FIFO range filter control register	R/W	0x0000	26.5.2.64/26-63
0x009C	LDTXSLAR—Last dynamic transmit slot channel A register	R	0x0000	26.5.2.65/26-64
0x009E	LDTXSLBR—Last dynamic transmit slot channel B register	R	0x0000	26.5.2.66/26-64
0x00A0	PCR0—Protocol configuration register 0	R/W	0x0000	26.5.2.67.1/26-67
0x00A2	PCR1—Protocol configuration register 1	R/W	0x0000	26.5.2.67.2/26-67
0x00A4	PCR2—Protocol configuration register 2	R/W	0x0000	26.5.2.67.3/26-67
0x00A6	PCR3—Protocol configuration register 3	R/W	0x0000	26.5.2.67.4/26-67
0x00A8	PCR4—Protocol configuration register 4	R/W	0x0000	26.5.2.67.5/26-67
0x00AA	PCR5—Protocol configuration register 5	R/W	0x0000	26.5.2.67.6/26-68
0x00AC	PCR6—Protocol configuration register 6	R/W	0x0000	26.5.2.67.7/26-68
0x00AE	PCR7—Protocol configuration register 7	R/W	0x0000	26.5.2.67.8/26-68
0x00B0	PCR8—Protocol configuration register 8	R/W	0x0000	26.5.2.67.9/26-68
0x00B2	PCR9—Protocol configuration register 9	R/W	0x0000	26.5.2.67.10/26-68
0x00B4	PCR10—Protocol configuration register 10	R/W	0x0000	26.5.2.67.11/26-69
0x00B6	PCR11—Protocol configuration register 11	R/W	0x0000	26.5.2.67.12/26-69
0x00B8	PCR12—Protocol configuration register 12	R/W	0x0000	26.5.2.67.13/26-69
0x00BA	PCR13—Protocol configuration register 13	R/W	0x0000	26.5.2.67.14/26-69
0x00BC	PCR14—Protocol configuration register 14	R/W	0x0000	26.5.2.67.15/26-70

Table A-4. PXN20 Detailed Register Map (continued)

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
0x00BE	PCR15—Protocol configuration register 15	R/W	0x0000	26.5.2.67.16/26-70
0x00C0	PCR16—Protocol configuration register 16	R/W	0x0000	26.5.2.67.17/26-70
0x00C2	PCR17—Protocol configuration register 17	R/W	0x0000	26.5.2.67.18/26-70
0x00C4	PCR18—Protocol configuration register 18	R/W	0x0000	26.5.2.67.19/26-70
0x00C6	PCR19—Protocol configuration register 19	R/W	0x0000	26.5.2.67.20/26-71
0x00C8	PCR20—Protocol configuration register 20	R/W	0x0000	26.5.2.67.21/26-71
0x00CA	PCR21—Protocol configuration register 21	R/W	0x0000	26.5.2.67.22/26-71
0x00CC	PCR22—Protocol configuration register 22	R/W	0x0000	26.5.2.67.23/26-71
0x00CE	PCR23—Protocol configuration register 23	R/W	0x0000	26.5.2.67.24/26-71
0x00D0	PCR24—Protocol configuration register 24	R/W	0x0000	26.5.2.67.25/26-72
0x00D2	PCR25—Protocol configuration register 25	R/W	0x0000	26.5.2.67.26/26-72
0x00D4	PCR26—Protocol configuration register 26	R/W	0x0000	26.5.2.67.27/26-72
0x00D6	PCR27—Protocol configuration register 27	R/W	0x0000	26.5.2.67.28/26-72
0x00D8	PCR28—Protocol configuration register 28	R/W	0x0000	26.5.2.67.29/26-72
0x00DA	PCR29—Protocol configuration register 29	R/W	0x0000	26.5.2.67.30/26-73
0x00DC	PCR30—Protocol configuration register 30	R/W	0x0000	26.5.2.67.31/26-73
0x00DE–0x00E6	Reserved			
0x00E8	RFSYMBADHR—Receive FIFO system memory base address high register	R/W	0x0000	26.5.2.51/26-56
0x00EA	RFSYMBADLR—Receive FIFO system memory base address low register	R/W	0x0000	26.5.2.51/26-56
0x00EC	RFPTR—Receive FIFO periodic timer register	R/W	0x0000	26.5.2.52/26-57
0x00EE	RFFLPCR—Receive FIFO fill level and pop count register	R/W	0x0000	26.5.2.58/26-60

Table A-4. PXN20 Detailed Register Map (continued)

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
0x00F0–0x00FF	Reserved			
0x0100	MBCCSR0–Message buffer configuration, control, status register 0	R/W	0x0000	26.5.2.68/26-73
0x0102	MBCCFR0–Message buffer cycle counter filter register 0	R/W	—[3]	26.5.2.69/26-75
0x0104	MBFIDR0–Message buffer frame ID register 0	R/W	0x0UUU	26.5.2.70/26-76
0x0106	MBIDXR0–Message buffer index register 0	R/W	0x00UU	26.5.2.71/26-76
0x0108	MBCCSR1–Message buffer configuration, control, status register 1	R/W	0x0000	26.5.2.68/26-73
0x010A	MBCCFR1–Message buffer cycle counter filter register 1	R/W	—[3]	26.5.2.69/26-75
0x010C	MBFIDR1–Message buffer frame ID register 1	R/W	0x0UUU	26.5.2.70/26-76
0x010E	MBIDXR1–Message buffer index register 1	R/W	0x00UU	26.5.2.71/26-76
0x0110	MBCCSR2–Message buffer configuration, control, status register 2	R/W	0x0000	26.5.2.68/26-73
0x0112	MBCCFR2–Message buffer cycle counter filter register 2	R/W	—[3]	26.5.2.69/26-75
0x0114	MBFIDR2–Message buffer frame ID register 2	R/W	0x0UUU	26.5.2.70/26-76
0x0116	MBIDXR2–Message buffer index register 2	R/W	0x00UU	26.5.2.71/26-76
0x0118	MBCCSR3–Message buffer configuration, control, status register 3	R/W	0x0000	26.5.2.68/26-73
0x011A	MBCCFR3–Message buffer cycle counter filter register 3	R/W	—[3]	26.5.2.69/26-75
0x011C	MBFIDR3–Message buffer frame ID register 3	R/W	0x0UUU	26.5.2.70/26-76
0x011E	MBIDXR3–Message buffer index register 3	R/W	0x00UU	26.5.2.71/26-76
0x0120	MBCCSR4–Message buffer configuration, control, status register 4	R/W	0x0000	26.5.2.68/26-73
0x0122	MBCCFR4–Message buffer cycle counter filter register 4	R/W	—[3]	26.5.2.69/26-75
0x0124	MBFIDR4–Message buffer frame ID register 4	R/W	0x0UUU	26.5.2.70/26-76
0x0126	MBIDXR4–Message buffer index register 4	R/W	0x00UU	26.5.2.71/26-76
0x0128	MBCCSR5–Message buffer configuration, control, status register 5	R/W	0x0000	26.5.2.68/26-73
0x012A	MBCCFR5–Message buffer cycle counter filter register 5	R/W	—[3]	26.5.2.69/26-75
0x012C	MBFIDR5–Message buffer frame ID register 5	R/W	0x0UUU	26.5.2.70/26-76
0x012E	MBIDXR5–Message buffer index register 5	R/W	0x00UU	26.5.2.71/26-76
0x0130	MBCCSR6–Message buffer configuration, control, status register 6	R/W	0x0000	26.5.2.68/26-73
0x0132	MBCCFR6–Message buffer cycle counter filter register 6	R/W	—[3]	26.5.2.69/26-75
0x0134	MBFIDR6–Message buffer frame ID register 6	R/W	0x0UUU	26.5.2.70/26-76
0x0136	MBIDXR6–Message buffer index register 6	R/W	0x00UU	26.5.2.71/26-76
0x0138	MBCCSR7–Message buffer configuration, control, status register 7	R/W	0x0000	26.5.2.68/26-73
0x013A	MBCCFR7–Message buffer cycle counter filter register 7	R/W	—[3]	26.5.2.69/26-75
0x013C	MBFIDR7–Message buffer frame ID register 7	R/W	0x0UUU	26.5.2.70/26-76

Table A-4. PXN20 Detailed Register Map (continued)

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
0x013E	MBIDXR7–Message buffer index register 7	R/W	0x00UU	26.5.2.71/26-76
0x0140	MBCCSR8–Message buffer configuration, control, status register 8	R/W	0x0000	26.5.2.68/26-73
0x0142	MBCCFR8–Message buffer cycle counter filter register 8	R/W	—[3]	26.5.2.69/26-75
0x0144	MBFIDR8–Message buffer frame ID register 8	R/W	0x0UUU	26.5.2.70/26-76
0x0146	MBIDXR8–Message buffer index register 8	R/W	0x00UU	26.5.2.71/26-76
0x0148	MBCCSR9–Message buffer configuration, control, status register 9	R/W	0x0000	26.5.2.68/26-73
0x014A	MBCCFR9–Message buffer cycle counter filter register 9	R/W	—[3]	26.5.2.69/26-75
0x014C	MBFIDR9–Message buffer frame ID register 9	R/W	0x0UUU	26.5.2.70/26-76
0x014E	MBIDXR9–Message buffer index register 9	R/W	0x00UU	26.5.2.71/26-76
0x0150	MBCCSR10–Message buffer configuration, control, status register 10	R/W	0x0000	26.5.2.68/26-73
0x0152	MBCCFR10–Message buffer cycle counter filter register 10	R/W	—[3]	26.5.2.69/26-75
0x0154	MBFIDR10–Message buffer frame ID register 10	R/W	0x0UUU	26.5.2.70/26-76
0x0156	MBIDXR10–Message buffer index register 10	R/W	0x00UU	26.5.2.71/26-76
0x0158	MBCCSR11–Message buffer configuration, control, status register 11	R/W	0x0000	26.5.2.68/26-73
0x015A	MBCCFR11–Message buffer cycle counter filter register 11	R/W	—[3]	26.5.2.69/26-75
0x015C	MBFIDR11–Message buffer frame ID register 11	R/W	0x0UUU	26.5.2.70/26-76
0x015E	MBIDXR11–Message buffer index register 11	R/W	0x00UU	26.5.2.71/26-76
0x0160	MBCCSR12–Message buffer configuration, control, status register 12	R/W	0x0000	26.5.2.68/26-73
0x0162	MBCCFR12–Message buffer cycle counter filter register 12	R/W	—[3]	26.5.2.69/26-75
0x0164	MBFIDR12–Message buffer frame ID register 12	R/W	0x0UUU	26.5.2.70/26-76
0x0166	MBIDXR12–Message buffer index register 12	R/W	0x00UU	26.5.2.71/26-76
0x0168	MBCCSR13–Message buffer configuration, control, status register 13	R/W	0x0000	26.5.2.68/26-73
0x016A	MBCCFR13–Message buffer cycle counter filter register 13	R/W	—[3]	26.5.2.69/26-75
0x016C	MBFIDR13–Message buffer frame ID register 13	R/W	0x0UUU	26.5.2.70/26-76
0x016E	MBIDXR13–Message buffer index register 13	R/W	0x00UU	26.5.2.71/26-76
0x0170	MBCCSR14–Message buffer configuration, control, status register 14	R/W	0x0000	26.5.2.68/26-73
0x0172	MBCCFR14–Message buffer cycle counter filter register 14	R/W	—[3]	26.5.2.69/26-75
0x0174	MBFIDR14–Message buffer frame ID register 14	R/W	0x0UUU	26.5.2.70/26-76
0x0176	MBIDXR14–Message buffer index register 14	R/W	0x00UU	26.5.2.71/26-76

Table A-4. PXN20 Detailed Register Map (continued)

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
0x0178	MBCCSR15–Message buffer configuration, control, status register 15	R/W	0x0000	26.5.2.68/26-73
0x017A	MBCCFR15–Message buffer cycle counter filter register 15	R/W	—[3]	26.5.2.69/26-75
0x017C	MBFIDR15–Message buffer frame ID register 15	R/W	0x0UUU	26.5.2.70/26-76
0x017E	MBIDXR15–Message buffer index register 15	R/W	0x00UU	26.5.2.71/26-76
0x0180	MBCCSR16–Message buffer configuration, control, status register 16	R/W	0x0000	26.5.2.68/26-73
0x0182	MBCCFR16–Message buffer cycle counter filter register 16	R/W	—[3]	26.5.2.69/26-75
0x0184	MBFIDR16–Message buffer frame ID register 16	R/W	0x0UUU	26.5.2.70/26-76
0x0186	MBIDXR16–Message buffer index register 16	R/W	0x00UU	26.5.2.71/26-76
0x0188	MBCCSR17–Message buffer configuration, control, status register 17	R/W	0x0000	26.5.2.68/26-73
0x018A	MBCCFR17–Message buffer cycle counter filter register 17	R/W	—[3]	26.5.2.69/26-75
0x018C	MBFIDR17–Message buffer frame ID register 17	R/W	0x0UUU	26.5.2.70/26-76
0x018E	MBIDXR17–Message buffer index register 17	R/W	0x00UU	26.5.2.71/26-76
0x0190	MBCCSR18–Message buffer configuration, control, status register 18	R/W	0x0000	26.5.2.68/26-73
0x0192	MBCCFR18–Message buffer cycle counter filter register 18	R/W	—[3]	26.5.2.69/26-75
0x0194	MBFIDR18–Message buffer frame ID register 18	R/W	0x0UUU	26.5.2.70/26-76
0x0196	MBIDXR18–Message buffer index register 18	R/W	0x00UU	26.5.2.71/26-76
0x0198	MBCCSR19–Message buffer configuration, control, status register 19	R/W	0x0000	26.5.2.68/26-73
0x019A	MBCCFR19–Message buffer cycle counter filter register 19	R/W	—[3]	26.5.2.69/26-75
0x019C	MBFIDR19–Message buffer frame ID register 19	R/W	0x0UUU	26.5.2.70/26-76
0x019E	MBIDXR19–Message buffer index register 19	R/W	0x00UU	26.5.2.71/26-76
0x01A0	MBCCSR20–Message buffer configuration, control, status register 20	R/W	0x0000	26.5.2.68/26-73
0x01A2	MBCCFR20–Message buffer cycle counter filter register 20	R/W	—[3]	26.5.2.69/26-75
0x01A4	MBFIDR20–Message buffer frame ID register 20	R/W	0x0UUU	26.5.2.70/26-76
0x01A6	MBIDXR20–Message buffer index register 20	R/W	0x00UU	26.5.2.71/26-76
0x01A8	MBCCSR21–Message buffer configuration, control, status register 21	R/W	0x0000	26.5.2.68/26-73
0x01AA	MBCCFR21–Message buffer cycle counter filter register 21	R/W	—[3]	26.5.2.69/26-75
0x01AC	MBFIDR21–Message buffer frame ID register 21	R/W	0x0UUU	26.5.2.70/26-76
0x01AE	MBIDXR21–Message buffer index register 21	R/W	0x00UU	26.5.2.71/26-76

Table A-4. PXN20 Detailed Register Map (continued)

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
0x01B0	MBCCSR22–Message buffer configuration, control, status register 22	R/W	0x0000	26.5.2.68/26-73
0x01B2	MBCCFR22–Message buffer cycle counter filter register 22	R/W	—[3]	26.5.2.69/26-75
0x01B4	MBFIDR22–Message buffer frame ID register 22	R/W	0x0UUU	26.5.2.70/26-76
0x01B6	MBIDXR22–Message buffer index register 22	R/W	0x00UU	26.5.2.71/26-76
0x01B8	MBCCSR23–Message buffer configuration, control, status register 23	R/W	0x0000	26.5.2.68/26-73
0x01BA	MBCCFR23–Message buffer cycle counter filter register 23	R/W	—[3]	26.5.2.69/26-75
0x01BC	MBFIDR23–Message buffer frame ID register 23	R/W	0x0UUU	26.5.2.70/26-76
0x01BE	MBIDXR23–Message buffer index register 23	R/W	0x00UU	26.5.2.71/26-76
0x01C0	MBCCSR24–Message buffer configuration, control, status register 24	R/W	0x0000	26.5.2.68/26-73
0x01C2	MBCCFR24–Message buffer cycle counter filter register 24	R/W	—[3]	26.5.2.69/26-75
0x01C4	MBFIDR24–Message buffer frame ID register 24	R/W	0x0UUU	26.5.2.70/26-76
0x01C6	MBIDXR24–Message buffer index register 24	R/W	0x00UU	26.5.2.71/26-76
0x01C8	MBCCSR25–Message buffer configuration, control, status register 25	R/W	0x0000	26.5.2.68/26-73
0x01CA	MBCCFR25–Message buffer cycle counter filter register 25	R/W	—[3]	26.5.2.69/26-75
0x01CC	MBFIDR25–Message buffer frame ID register 25	R/W	0x0UUU	26.5.2.70/26-76
0x01CE	MBIDXR25–Message buffer index register 25	R/W	0x00UU	26.5.2.71/26-76
0x01D0	MBCCSR26–Message buffer configuration, control, status register 26	R/W	0x0000	26.5.2.68/26-73
0x01D2	MBCCFR26–Message buffer cycle counter filter register 26	R/W	—[3]	26.5.2.69/26-75
0x01D4	MBFIDR26–Message buffer frame ID register 26	R/W	0x0UUU	26.5.2.70/26-76
0x01D6	MBIDXR26–Message buffer index register 26	R/W	0x00UU	26.5.2.71/26-76
0x01D8	MBCCSR27–Message buffer configuration, control, status register 27	R/W	0x0000	26.5.2.68/26-73
0x01DA	MBCCFR27–Message buffer cycle counter filter register 27	R/W	—[3]	26.5.2.69/26-75
0x01DC	MBFIDR27–Message buffer frame ID register 27	R/W	0x0UUU	26.5.2.70/26-76
0x01DE	MBIDXR27–Message buffer index register 27	R/W	0x00UU	26.5.2.71/26-76
0x01E0	MBCCSR28–Message buffer configuration, control, status register 28	R/W	0x0000	26.5.2.68/26-73
0x01E2	MBCCFR28–Message buffer cycle counter filter register28	R/W	—[3]	26.5.2.69/26-75
0x01E4	MBFIDR28–Message buffer frame ID register 28	R/W	0x0UUU	26.5.2.70/26-76
0x01E6	MBIDXR28–Message buffer index register 28	R/W	0x00UU	26.5.2.71/26-76

Table A-4. PXN20 Detailed Register Map (continued)

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
0x01E8	MBCCSR29–Message buffer configuration, control, status register 29	R/W	0x0000	26.5.2.68/26-73
0x01EA	MBCCFR29–Message buffer cycle counter filter register 29	R/W	—[3]	26.5.2.69/26-75
0x01EC	MBFIDR29–Message buffer frame ID register 29	R/W	0x0UUU	26.5.2.70/26-76
0x01EE	MBIDXR29–Message buffer index register 29	R/W	0x00UU	26.5.2.71/26-76
0x01F0	MBCCSR30–Message buffer configuration, control, status register 30	R/W	0x0000	26.5.2.68/26-73
0x01F2	MBCCFR30–Message buffer cycle counter filter register 30	R/W	—[3]	26.5.2.69/26-75
0x01F4	MBFIDR30–Message buffer frame ID register 30	R/W	0x0UUU	26.5.2.70/26-76
0x01F6	MBIDXR30–Message buffer index register 30	R/W	0x00UU	26.5.2.71/26-76
0x01F8	MBCCSR31–Message buffer configuration, control, status register 31	R/W	0x0000	26.5.2.68/26-73
0x01FA	MBCCFR31–Message buffer cycle counter filter register 31	R/W	—[3]	26.5.2.69/26-75
0x01FC	MBFIDR31–Message buffer frame ID register 31	R/W	0x0UUU	26.5.2.70/26-76
0x01FE	MBIDXR31–Message buffer index register 31	R/W	0x00UU	26.5.2.71/26-76
0x0200	MBCCSR32–Message buffer configuration, control, status register 32	R/W	0x0000	26.5.2.68/26-73
0x0202	MBCCFR32–Message buffer cycle counter filter register 32	R/W	—[3]	26.5.2.69/26-75
0x0204	MBFIDR32–Message buffer frame ID register 32	R/W	0x0UUU	26.5.2.70/26-76
0x0206	MBIDXR32–Message buffer index register 32	R/W	0x00UU	26.5.2.71/26-76
0x0208	MBCCSR33–Message buffer configuration, control, status register 33	R/W	0x0000	26.5.2.68/26-73
0x020A	MBCCFR33–Message buffer cycle counter filter register33	R/W	—[3]	26.5.2.69/26-75
0x020C	MBFIDR33–Message buffer frame ID register 33	R/W	0x0UUU	26.5.2.70/26-76
0x020E	MBIDXR33–Message buffer index register 33	R/W	0x00UU	26.5.2.71/26-76
0x0210	MBCCSR34–Message buffer configuration, control, status register 34	R/W	0x0000	26.5.2.68/26-73
0x0212	MBCCFR34–Message buffer cycle counter filter register 34	R/W	—[3]	26.5.2.69/26-75
0x0214	MBFIDR34–Message buffer frame ID register 34	R/W	0x0UUU	26.5.2.70/26-76
0x0216	MBIDXR34–Message buffer index register 34	R/W	0x00UU	26.5.2.71/26-76
0x0218	MBCCSR35–Message buffer configuration, control, status register 35	R/W	0x0000	26.5.2.68/26-73
0x021A	MBCCFR35–Message buffer cycle counter filter register 35	R/W	—[3]	26.5.2.69/26-75
0x021C	MBFIDR35–Message buffer frame ID register 35	R/W	0x0UUU	26.5.2.70/26-76
0x021E	MBIDXR35–Message buffer index register 35	R/W	0x00UU	26.5.2.71/26-76

Table A-4. PXN20 Detailed Register Map (continued)

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
0x0220	MBCCSR36–Message buffer configuration, control, status register 36	R/W	0x0000	26.5.2.68/26-73
0x0222	MBCCFR36–Message buffer cycle counter filter register 36	R/W	—[3]	26.5.2.69/26-75
0x0224	MBFIDR36–Message buffer frame ID register 36	R/W	0x0UUU	26.5.2.70/26-76
0x0226	MBIDXR36–Message buffer index register 36	R/W	0x00UU	26.5.2.71/26-76
0x0228	MBCCSR37–Message buffer configuration, control, status register 37	R/W	0x0000	26.5.2.68/26-73
0x022A	MBCCFR37–Message buffer cycle counter filter register 37	R/W	—[3]	26.5.2.69/26-75
0x022C	MBFIDR37–Message buffer frame ID register 37	R/W	0x0UUU	26.5.2.70/26-76
0x022E	MBIDXR37–Message buffer index register 37	R/W	0x00UU	26.5.2.71/26-76
0x0230	MBCCSR38–Message buffer configuration, control, status register 38	R/W	0x0000	26.5.2.68/26-73
0x0232	MBCCFR38–Message buffer cycle counter filter register 38	R/W	—[3]	26.5.2.69/26-75
0x0234	MBFIDR38–Message buffer frame ID register 38	R/W	0x0UUU	26.5.2.70/26-76
0x0236	MBIDXR38–Message buffer index register 38	R/W	0x00UU	26.5.2.71/26-76
0x0238	MBCCSR39–Message buffer configuration, control, status register 39	R/W	0x0000	26.5.2.68/26-73
0x023A	MBCCFR39–Message buffer cycle counter filter register 39	R/W	—[3]	26.5.2.69/26-75
0x023C	MBFIDR39–Message buffer frame ID register 39	R/W	0x0UUU	26.5.2.70/26-76
0x023E	MBIDXR39–Message buffer index register 39	R/W	0x00UU	26.5.2.71/26-76
0x0240	MBCCSR40–Message buffer configuration, control, status register 40	R/W	0x0000	26.5.2.68/26-73
0x0242	MBCCFR40–Message buffer cycle counter filter register40	R/W	—[3]	26.5.2.69/26-75
0x0244	MBFIDR40–Message buffer frame ID register 40	R/W	0x0UUU	26.5.2.70/26-76
0x0246	MBIDXR40–Message buffer index register 40	R/W	0x00UU	26.5.2.71/26-76
0x0248	MBCCSR41–Message buffer configuration, control, status register 41	R/W	0x0000	26.5.2.68/26-73
0x024A	MBCCFR41–Message buffer cycle counter filter register 41	R/W	—[3]	26.5.2.69/26-75
0x024C	MBFIDR41–Message buffer frame ID register 41	R/W	0x0UUU	26.5.2.70/26-76
0x024E	MBIDXR41–Message buffer index register 41	R/W	0x00UU	26.5.2.71/26-76
0x0250	MBCCSR42–Message buffer configuration, control, status register 42	R/W	0x0000	26.5.2.68/26-73
0x0252	MBCCFR42–Message buffer cycle counter filter register 42	R/W	—[3]	26.5.2.69/26-75
0x0254	MBFIDR42–Message buffer frame ID register 42	R/W	0x0UUU	26.5.2.70/26-76
0x0256	MBIDXR42–Message buffer index register 42	R/W	0x00UU	26.5.2.71/26-76

Table A-4. PXN20 Detailed Register Map (continued)

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
0x0258	MBCCSR43–Message buffer configuration, control, status register 43	R/W	0x0000	26.5.2.68/26-73
0x025A	MBCCFR43–Message buffer cycle counter filter register 43	R/W	—[3]	26.5.2.69/26-75
0x025C	MBFIDR43–Message buffer frame ID register 43	R/W	0x0UUU	26.5.2.70/26-76
0x025E	MBIDXR43–Message buffer index register 43	R/W	0x00UU	26.5.2.71/26-76
0x0260	MBCCSR44–Message buffer configuration, control, status register 44	R/W	0x0000	26.5.2.68/26-73
0x0262	MBCCFR44–Message buffer cycle counter filter register 44	R/W	—[3]	26.5.2.69/26-75
0x0264	MBFIDR44–Message buffer frame ID register 44	R/W	0x0UUU	26.5.2.70/26-76
0x0266	MBIDXR44–Message buffer index register 44	R/W	0x00UU	26.5.2.71/26-76
0x0268	MBCCSR45–Message buffer configuration, control, status register 45	R/W	0x0000	26.5.2.68/26-73
0x026A	MBCCFR45–Message buffer cycle counter filter register45	R/W	—[3]	26.5.2.69/26-75
0x026C	MBFIDR45–Message buffer frame ID register 45	R/W	0x0UUU	26.5.2.70/26-76
0x026E	MBIDXR45–Message buffer index register 45	R/W	0x00UU	26.5.2.71/26-76
0x0270	MBCCSR46–Message buffer configuration, control, status register 46	R/W	0x0000	26.5.2.68/26-73
0x0272	MBCCFR46–Message buffer cycle counter filter register 46	R/W	—[3]	26.5.2.69/26-75
0x0274	MBFIDR46–Message buffer frame ID register 46	R/W	0x0UUU	26.5.2.70/26-76
0x0276	MBIDXR46–Message buffer index register 46	R/W	0x00UU	26.5.2.71/26-76
0x0278	MBCCSR47–Message buffer configuration, control, status register 47	R/W	0x0000	26.5.2.68/26-73
0x027A	MBCCFR47–Message buffer cycle counter filter register 47	R/W	—[3]	26.5.2.69/26-75
0x027C	MBFIDR47–Message buffer frame ID register 47	R/W	0x0UUU	26.5.2.70/26-76
0x027E	MBIDXR47–Message buffer index register 47	R/W	0x00UU	26.5.2.71/26-76
0x0280	MBCCSR48–Message buffer configuration, control, status register 48	R/W	0x0000	26.5.2.68/26-73
0x0282	MBCCFR48–Message buffer cycle counter filter register 48	R/W	—[3]	26.5.2.69/26-75
0x0284	MBFIDR48–Message buffer frame ID register 48	R/W	0x0UUU	26.5.2.70/26-76
0x0286	MBIDXR48–Message buffer index register 48	R/W	0x00UU	26.5.2.71/26-76
0x0288	MBCCSR49–Message buffer configuration, control, status register 49	R/W	0x0000	26.5.2.68/26-73
0x028A	MBCCFR49–Message buffer cycle counter filter register49	R/W	—[3]	26.5.2.69/26-75
0x028C	MBFIDR49–Message buffer frame ID register 49	R/W	0x0UUU	26.5.2.70/26-76
0x028E	MBIDXR49–Message buffer index register 49	R/W	0x00UU	26.5.2.71/26-76

Table A-4. PXN20 Detailed Register Map (continued)

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
0x0290	MBCCSR50–Message buffer configuration, control, status register 50	R/W	0x0000	26.5.2.68/26-73
0x0292	MBCCFR50–Message buffer cycle counter filter register 50	R/W	—[3]	26.5.2.69/26-75
0x0294	MBFIDR50–Message buffer frame ID register 50	R/W	0x0UUU	26.5.2.70/26-76
0x0296	MBIDXR50–Message buffer index register 50	R/W	0x00UU	26.5.2.71/26-76
0x0298	MBCCSR51–Message buffer configuration, control, status register 51	R/W	0x0000	26.5.2.68/26-73
0x029A	MBCCFR51–Message buffer cycle counter filter register 51	R/W	—[3]	26.5.2.69/26-75
0x029C	MBFIDR51–Message buffer frame ID register 51	R/W	0x0UUU	26.5.2.70/26-76
0x029E	MBIDXR51–Message buffer index register 51	R/W	0x00UU	26.5.2.71/26-76
0x02A0	MBCCSR52–Message buffer configuration, control, status register 52	R/W	0x0000	26.5.2.68/26-73
0x02A2	MBCCFR52–Message buffer cycle counter filter register 52	R/W	—[3]	26.5.2.69/26-75
0x02A4	MBFIDR52–Message buffer frame ID register 52	R/W	0x0UUU	26.5.2.70/26-76
0x02A6	MBIDXR52–Message buffer index register 52	R/W	0x00UU	26.5.2.71/26-76
0x02A8	MBCCSR53–Message buffer configuration, control, status register 53	R/W	0x0000	26.5.2.68/26-73
0x02AA	MBCCFR53–Message buffer cycle counter filter register 53	R/W	—[3]	26.5.2.69/26-75
0x02AC	MBFIDR53–Message buffer frame ID register 53	R/W	0x0UUU	26.5.2.70/26-76
0x02AE	MBIDXR53–Message buffer index register 53	R/W	0x00UU	26.5.2.71/26-76
0x02B0	MBCCSR54–Message buffer configuration, control, status register 54	R/W	0x0000	26.5.2.68/26-73
0x02B2	MBCCFR54–Message buffer cycle counter filter register 54	R/W	—[3]	26.5.2.69/26-75
0x02B4	MBFIDR54–Message buffer frame ID register 54	R/W	0x0UUU	26.5.2.70/26-76
0x02B6	MBIDXR54–Message buffer index register 54	R/W	0x00UU	26.5.2.71/26-76
0x02B8	MBCCSR55–Message buffer configuration, control, status register 55	R/W	0x0000	26.5.2.68/26-73
0x02BA	MBCCFR55–Message buffer cycle counter filter register 55	R/W	—[3]	26.5.2.69/26-75
0x02BC	MBFIDR55–Message buffer frame ID register 55	R/W	0x0UUU	26.5.2.70/26-76
0x02BE	MBIDXR55–Message buffer index register 55	R/W	0x00UU	26.5.2.71/26-76
0x02C0	MBCCSR56–Message buffer configuration, control, status register 56	R/W	0x0000	26.5.2.68/26-73
0x02C2	MBCCFR56–Message buffer cycle counter filter register 56	R/W	—[3]	26.5.2.69/26-75
0x02C4	MBFIDR56–Message buffer frame ID register 56	R/W	0x0UUU	26.5.2.70/26-76
0x02C6	MBIDXR56–Message buffer index register 56	R/W	0x00UU	26.5.2.71/26-76

Table A-4. PXN20 Detailed Register Map (continued)

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
0x02C8	MBCCSR57–Message buffer configuration, control, status register 57	R/W	0x0000	26.5.2.68/26-73
0x02CA	MBCCFR57–Message buffer cycle counter filter register57	R/W	—[3]	26.5.2.69/26-75
0x02CC	MBFIDR57–Message buffer frame ID register 57	R/W	0x0UUU	26.5.2.70/26-76
0x02CE	MBIDXR57–Message buffer index register 57	R/W	0x00UU	26.5.2.71/26-76
0x02D0	MBCCSR58–Message buffer configuration, control, status register 58	R/W	0x0000	26.5.2.68/26-73
0x02D2	MBCCFR58–Message buffer cycle counter filter register 58	R/W	—[3]	26.5.2.69/26-75
0x02D4	MBFIDR58–Message buffer frame ID register 58	R/W	0x0UUU	26.5.2.70/26-76
0x02D6	MBIDXR58–Message buffer index register 58	R/W	0x00UU	26.5.2.71/26-76
0x02D8	MBCCSR59–Message buffer configuration, control, status register 59	R/W	0x0000	26.5.2.68/26-73
0x02DA	MBCCFR59–Message buffer cycle counter filter register 59	R/W	—[3]	26.5.2.69/26-75
0x02DC	MBFIDR59–Message buffer frame ID register 59	R/W	0x0UUU	26.5.2.70/26-76
0x02DE	MBIDXR59–Message buffer index register 59	R/W	0x00UU	26.5.2.71/26-76
0x02E0	MBCCSR60–Message buffer configuration, control, status register 60	R/W	0x0000	26.5.2.68/26-73
0x02E2	MBCCFR60–Message buffer cycle counter filter register 60	R/W	—[3]	26.5.2.69/26-75
0x02E4	MBFIDR60–Message buffer frame ID register 60	R/W	0x0UUU	26.5.2.70/26-76
0x02E6	MBIDXR60–Message buffer index register 60	R/W	0x00UU	26.5.2.71/26-76
0x02E8	MBCCSR61–Message buffer configuration, control, status register 61	R/W	0x0000	26.5.2.68/26-73
0x02EA	MBCCFR61–Message buffer cycle counter filter register 61	R/W	—[3]	26.5.2.69/26-75
0x02EC	MBFIDR61–Message buffer frame ID register 61	R/W	0x0UUU	26.5.2.70/26-76
0x02EE	MBIDXR61–Message buffer index register 61	R/W	0x00UU	26.5.2.71/26-76
0x02F0	MBCCSR62–Message buffer configuration, control, status register 62	R/W	0x0000	26.5.2.68/26-73
0x02F2	MBCCFR62–Message buffer cycle counter filter register 62	R/W	—[3]	26.5.2.69/26-75
0x02F4	MBFIDR62–Message buffer frame ID register 62	R/W	0x0UUU	26.5.2.70/26-76
0x02F6	MBIDXR62–Message buffer index register 62	R/W	0x00UU	26.5.2.71/26-76
0x02F8	MBCCSR63–Message buffer configuration, control, status register 63	R/W	0x0000	26.5.2.68/26-73
0x02FA	MBCCFR63–Message buffer cycle counter filter register 63	R/W	—[3]	26.5.2.69/26-75
0x02FC	MBFIDR63–Message buffer frame ID register 63	R/W	0x0UUU	26.5.2.70/26-76
0x02FE	MBIDXR63–Message buffer index register 63	R/W	0x00UU	26.5.2.71/26-76

Table A-4. PXN20 Detailed Register Map (continued)

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
0x0300	MBCCSR64–Message buffer configuration, control, status register 64	R/W	0x0000	26.5.2.68/26-73
0x0302	MBCCFR64–Message buffer cycle counter filter register 64	R/W	—[3]	26.5.2.69/26-75
0x0304	MBFIDR64–Message buffer frame ID register 64	R/W	0x0UUU	26.5.2.70/26-76
0x0306	MBIDXR64–Message buffer index register 64	R/W	0x00UU	26.5.2.71/26-76
0x0308	MBCCSR65–Message buffer configuration, control, status register 65	R/W	0x0000	26.5.2.68/26-73
0x030A	MBCCFR65–Message buffer cycle counter filter register 65	R/W	—[3]	26.5.2.69/26-75
0x030C	MBFIDR65–Message buffer frame ID register 65	R/W	0x0UUU	26.5.2.70/26-76
0x030E	MBIDXR65–Message buffer index register 65	R/W	0x00UU	26.5.2.71/26-76
0x0310	MBCCSR66–Message buffer configuration, control, status register 66	R/W	0x0000	26.5.2.68/26-73
0x0312	MBCCFR66–Message buffer cycle counter filter register 66	R/W	—[3]	26.5.2.69/26-75
0x0314	MBFIDR66–Message buffer frame ID register 66	R/W	0x0UUU	26.5.2.70/26-76
0x0316	MBIDXR66–Message buffer index register 66	R/W	0x00UU	26.5.2.71/26-76
0x0318	MBCCSR67–Message buffer configuration, control, status register 67	R/W	0x0000	26.5.2.68/26-73
0x031A	MBCCFR67–Message buffer cycle counter filter register 67	R/W	—[3]	26.5.2.69/26-75
0x031C	MBFIDR67–Message buffer frame ID register 67	R/W	0x0UUU	26.5.2.70/26-76
0x031E	MBIDXR67–Message buffer index register 67	R/W	0x00UU	26.5.2.71/26-76
0x0320	MBCCSR68–Message buffer configuration, control, status register 68	R/W	0x0000	26.5.2.68/26-73
0x0322	MBCCFR68–Message buffer cycle counter filter register 68	R/W	—[3]	26.5.2.69/26-75
0x0324	MBFIDR68–Message buffer frame ID register 68	R/W	0x0UUU	26.5.2.70/26-76
0x0326	MBIDXR68–Message buffer index register 68	R/W	0x00UU	26.5.2.71/26-76
0x0328	MBCCSR69–Message buffer configuration, control, status register 69	R/W	0x0000	26.5.2.68/26-73
0x032A	MBCCFR69–Message buffer cycle counter filter register 69	R/W	—[3]	26.5.2.69/26-75
0x032C	MBFIDR69–Message buffer frame ID register 69	R/W	0x0UUU	26.5.2.70/26-76
0x032E	MBIDXR69–Message buffer index register 69	R/W	0x00UU	26.5.2.71/26-76
0x0330	MBCCSR70–Message buffer configuration, control, status register 70	R/W	0x0000	26.5.2.68/26-73
0x0332	MBCCFR70–Message buffer cycle counter filter register 70	R/W	—[3]	26.5.2.69/26-75
0x0334	MBFIDR70–Message buffer frame ID register 70	R/W	0x0UUU	26.5.2.70/26-76
0x0336	MBIDXR70–Message buffer index register 70	R/W	0x00UU	26.5.2.71/26-76

Table A-4. PXN20 Detailed Register Map (continued)

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
0x0338	MBCCS71–Message buffer configuration, control, status register 71	R/W	0x0000	26.5.2.68/26-73
0x033A	MBCCFR71–Message buffer cycle counter filter register 71	R/W	—[3]	26.5.2.69/26-75
0x033C	MBFIDR71–Message buffer frame ID register 71	R/W	0x0UUU	26.5.2.70/26-76
0x033E	MBIDXR71–Message buffer index register 71	R/W	0x00UU	26.5.2.71/26-76
0x0340	MBCCSR72–Message buffer configuration, control, status register 72	R/W	0x0000	26.5.2.68/26-73
0x0342	MBCCFR72–Message buffer cycle counter filter register72	R/W	—[3]	26.5.2.69/26-75
0x0344	MBFIDR72–Message buffer frame ID register 72	R/W	0x0UUU	26.5.2.70/26-76
0x0346	MBIDXR72–Message buffer index register 72	R/W	0x00UU	26.5.2.71/26-76
0x0348	MBCCSR73–Message buffer configuration, control, status register 73	R/W	0x0000	26.5.2.68/26-73
0x034A	MBCCFR73–Message buffer cycle counter filter register 73	R/W	—[3]	26.5.2.69/26-75
0x034C	MBFIDR73–Message buffer frame ID register 73	R/W	0x0UUU	26.5.2.70/26-76
0x034E	MBIDXR73–Message buffer index register 73	R/W	0x00UU	26.5.2.71/26-76
0x0350	MBCCSR74–Message buffer configuration, control, status register 74	R/W	0x0000	26.5.2.68/26-73
0x0352	MBCCFR74–Message buffer cycle counter filter register 74	R/W	—[3]	26.5.2.69/26-75
0x0354	MBFIDR74–Message buffer frame ID register 74	R/W	0x0UUU	26.5.2.70/26-76
0x0356	MBIDXR74–Message buffer index register 74	R/W	0x00UU	26.5.2.71/26-76
0x0358	MBCCSR75–Message buffer configuration, control, status register 75	R/W	0x0000	26.5.2.68/26-73
0x035A	MBCCFR75–Message buffer cycle counter filter register 75	R/W	—[3]	26.5.2.69/26-75
0x035C	MBFIDR75–Message buffer frame ID register 75	R/W	0x0UUU	26.5.2.70/26-76
0x035E	MBIDXR75–Message buffer index register 75	R/W	0x00UU	26.5.2.71/26-76
0x0360	MBCCSR76–Message buffer configuration, control, status register 76	R/W	0x0000	26.5.2.68/26-73
0x0362	MBCCFR76–Message buffer cycle counter filter register 76	R/W	—[3]	26.5.2.69/26-75
0x0364	MBFIDR76–Message buffer frame ID register 76	R/W	0x0UUU	26.5.2.70/26-76
0x0366	MBIDXR76–Message buffer index register 76	R/W	0x00UU	26.5.2.71/26-76
0x0368	MBCCSR77–Message buffer configuration, control, status register 77	R/W	0x0000	26.5.2.68/26-73
0x036A	MBCCFR77–Message buffer cycle counter filter register 77	R/W	—[3]	26.5.2.69/26-75
0x036C	MBFIDR77–Message buffer frame ID register 77	R/W	0x0UUU	26.5.2.70/26-76
0x036E	MBIDXR77–Message buffer index register 77	R/W	0x00UU	26.5.2.71/26-76

Table A-4. PXN20 Detailed Register Map (continued)

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
0x0370	MBCCSR78–Message buffer configuration, control, status register 78	R/W	0x0000	26.5.2.68/26-73
0x0372	MBCCFR78–Message buffer cycle counter filter register 78	R/W	—[3]	26.5.2.69/26-75
0x0374	MBFIDR78–Message buffer frame ID register 78	R/W	0x0UUU	26.5.2.70/26-76
0x0376	MBIDXR78–Message buffer index register 78	R/W	0x00UU	26.5.2.71/26-76
0x0378	MBCCSR79–Message buffer configuration, control, status register 79	R/W	0x0000	26.5.2.68/26-73
0x037A	MBCCFR79–Message buffer cycle counter filter register 79	R/W	—[3]	26.5.2.69/26-75
0x037C	MBFIDR79–Message buffer frame ID register 79	R/W	0x0UUU	26.5.2.70/26-76
0x037E	MBIDXR79–Message buffer index register 79	R/W	0x00UU	26.5.2.71/26-76
0x0380	MBCCSR80–Message buffer configuration, control, status register 80	R/W	0x0000	26.5.2.68/26-73
0x0382	MBCCFR80–Message buffer cycle counter filter register 80	R/W	—[3]	26.5.2.69/26-75
0x0384	MBFIDR80–Message buffer frame ID register 80	R/W	0x0UUU	26.5.2.70/26-76
0x0386	MBIDXR80–Message buffer index register 80	R/W	0x00UU	26.5.2.71/26-76
0x0388	MBCCSR81–Message buffer configuration, control, status register 81	R/W	0x0000	26.5.2.68/26-73
0x038A	MBCCFR81–Message buffer cycle counter filter register 81	R/W	—[3]	26.5.2.69/26-75
0x038C	MBFIDR81–Message buffer frame ID register 81	R/W	0x0UUU	26.5.2.70/26-76
0x038E	MBIDXR81–Message buffer index register 81	R/W	0x00UU	26.5.2.71/26-76
0x0390	MBCCSR82–Message buffer configuration, control, status register 82	R/W	0x0000	26.5.2.68/26-73
0x0392	MBCCFR82–Message buffer cycle counter filter register 82	R/W	—[3]	26.5.2.69/26-75
0x0394	MBFIDR82–Message buffer frame ID register 82	R/W	0x0UUU	26.5.2.70/26-76
0x0396	MBIDXR82–Message buffer index register 82	R/W	0x00UU	26.5.2.71/26-76
0x0398	MBCCSR83–Message buffer configuration, control, status register 83	R/W	0x0000	26.5.2.68/26-73
0x039A	MBCCFR83–Message buffer cycle counter filter register 83	R/W	—[3]	26.5.2.69/26-75
0x039C	MBFIDR83–Message buffer frame ID register 83	R/W	0x0UUU	26.5.2.70/26-76
0x039E	MBIDXR83–Message buffer index register 83	R/W	0x00UU	26.5.2.71/26-76
0x03A0	MBCCSR84–Message buffer configuration, control, status register 84	R/W	0x0000	26.5.2.68/26-73
0x03A2	MBCCFR84–Message buffer cycle counter filter register84	R/W	—[3]	26.5.2.69/26-75
0x03A4	MBFIDR84–Message buffer frame ID register 84	R/W	0x0UUU	26.5.2.70/26-76
0x03A6	MBIDXR84–Message buffer index register 84	R/W	0x00UU	26.5.2.71/26-76

Table A-4. PXN20 Detailed Register Map (continued)

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
0x03A8	MBCCSR85–Message buffer configuration, control, status register 85	R/W	0x0000	26.5.2.68/26-73
0x03AA	MBCCFR85–Message buffer cycle counter filter register 85	R/W	—[3]	26.5.2.69/26-75
0x03AC	MBFIDR85–Message buffer frame ID register 85	R/W	0x0UUU	26.5.2.70/26-76
0x03AE	MBIDXR85–Message buffer index register 85	R/W	0x00UU	26.5.2.71/26-76
0x03B0	MBCCSR86–Message buffer configuration, control, status register 86	R/W	0x0000	26.5.2.68/26-73
0x03B2	MBCCFR86–Message buffer cycle counter filter register 86	R/W	—[3]	26.5.2.69/26-75
0x03B4	MBFIDR86–Message buffer frame ID register 86	R/W	0x0UUU	26.5.2.70/26-76
0x03B6	MBIDXR86–Message buffer index register 86	R/W	0x00UU	26.5.2.71/26-76
0x03B8	MBCCSR87–Message buffer configuration, control, status register 87	R/W	0x0000	26.5.2.68/26-73
0x03BA	MBCCFR87–Message buffer cycle counter filter register 87	R/W	—[3]	26.5.2.69/26-75
0x03BC	MBFIDR87–Message buffer frame ID register 87	R/W	0x0UUU	26.5.2.70/26-76
0x03BE	MBIDXR87–Message buffer index register 87	R/W	0x00UU	26.5.2.71/26-76
0x03C0	MBCCSR88–Message buffer configuration, control, status register 88	R/W	0x0000	26.5.2.68/26-73
0x03C2	MBCCFR88–Message buffer cycle counter filter register 88	R/W	—[3]	26.5.2.69/26-75
0x03C4	MBFIDR88–Message buffer frame ID register 88	R/W	0x0UUU	26.5.2.70/26-76
0x03C6	MBIDXR88–Message buffer index register 88	R/W	0x00UU	26.5.2.71/26-76
0x03C8	MBCCSR89–Message buffer configuration, control, status register 89	R/W	0x0000	26.5.2.68/26-73
0x03CA	MBCCFR89–Message buffer cycle counter filter register 89	R/W	—[3]	26.5.2.69/26-75
0x03CC	MBFIDR89–Message buffer frame ID register 89	R/W	0x0UUU	26.5.2.70/26-76
0x03CE	MBIDXR89–Message buffer index register 89	R/W	0x00UU	26.5.2.71/26-76
0x03D0	MBCCSR90–Message buffer configuration, control, status register 90	R/W	0x0000	26.5.2.68/26-73
0x03D2	MBCCFR90–Message buffer cycle counter filter register 90	R/W	—[3]	26.5.2.69/26-75
0x03D4	MBFIDR90–Message buffer frame ID register 90	R/W	0x0UUU	26.5.2.70/26-76
0x03D6	MBIDXR90–Message buffer index register 90	R/W	0x00UU	26.5.2.71/26-76
0x03D8	MBCCSR91–Message buffer configuration, control, status register 91	R/W	0x0000	26.5.2.68/26-73
0x03DA	MBCCFR91–Message buffer cycle counter filter register 91	R/W	—[3]	26.5.2.69/26-75
0x03DC	MBFIDR91–Message buffer frame ID register 91	R/W	0x0UUU	26.5.2.70/26-76
0x03DE	MBIDXR91–Message buffer index register 91	R/W	0x00UU	26.5.2.71/26-76

Table A-4. PXN20 Detailed Register Map (continued)

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
0x03E0	MBCCSR92–Message buffer configuration, control, status register 92	R/W	0x0000	26.5.2.68/26-73
0x03E2	MBCCFR92–Message buffer cycle counter filter register 92	R/W	—[3]	26.5.2.69/26-75
0x03E4	MBFIDR92–Message buffer frame ID register 92	R/W	0x0UUU	26.5.2.70/26-76
0x03E6	MBIDXR92–Message buffer index register 92	R/W	0x00UU	26.5.2.71/26-76
0x03E8	MBCCSR93–Message buffer configuration, control, status register 93	R/W	0x0000	26.5.2.68/26-73
0x03EA	MBCCFR93–Message buffer cycle counter filter register 93	R/W	—[3]	26.5.2.69/26-75
0x03EC	MBFIDR93–Message buffer frame ID register 93	R/W	0x0UUU	26.5.2.70/26-76
0x03EE	MBIDXR93–Message buffer index register 93	R/W	0x00UU	26.5.2.71/26-76
0x03F0	MBCCSR94–Message buffer configuration, control, status register 94	R/W	0x0000	26.5.2.68/26-73
0x03F2	MBCCFR94–Message buffer cycle counter filter register 94	R/W	—[3]	26.5.2.69/26-75
0x03F4	MBFIDR94–Message buffer frame ID register 94	R/W	0x0UUU	26.5.2.70/26-76
0x03F6	MBIDXR94–Message buffer index register 94	R/W	0x00UU	26.5.2.71/26-76
0x03F8	MBCCSR95–Message buffer configuration, control, status register 95	R/W	0x0000	26.5.2.68/26-73
0x03FA	MBCCFR95–Message buffer cycle counter filter register 95	R/W	—[3]	26.5.2.69/26-75
0x03FC	MBFIDR95–Message buffer frame ID register 95	R/W	0x0UUU	26.5.2.70/26-76
0x03FE	MBIDXR95–Message buffer index register 95	R/W	0x00UU	26.5.2.71/26-76
0x0400	MBCCSR96–Message buffer configuration, control, status register 96	R/W	0x0000	26.5.2.68/26-73
0x0402	MBCCFR96–Message buffer cycle counter filter register 96	R/W	—[3]	26.5.2.69/26-75
0x0404	MBFIDR96–Message buffer frame ID register 96	R/W	0x0UUU	26.5.2.70/26-76
0x0406	MBIDXR96–Message buffer index register 96	R/W	0x00UU	26.5.2.71/26-76
0x0408	MBCCSR97–Message buffer configuration, control, status register 97	R/W	0x0000	26.5.2.68/26-73
0x040A	MBCCFR97–Message buffer cycle counter filter register 97	R/W	—[3]	26.5.2.69/26-75
0x040C	MBFIDR97–Message buffer frame ID register 97	R/W	0x0UUU	26.5.2.70/26-76
0x040E	MBIDXR97–Message buffer index register 97	R/W	0x00UU	26.5.2.71/26-76
0x0410	MBCCSR98–Message buffer configuration, control, status register 98	R/W	0x0000	26.5.2.68/26-73
0x0412	MBCCFR98–Message buffer cycle counter filter register 98	R/W	—[3]	26.5.2.69/26-75
0x0414	MBFIDR98–Message buffer frame ID register 98	R/W	0x0UUU	26.5.2.70/26-76
0x0416	MBIDXR98–Message buffer index register 98	R/W	0x00UU	26.5.2.71/26-76

Table A-4. PXN20 Detailed Register Map (continued)

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
0x0418	MBCCSR99–Message buffer configuration, control, status register 99	R/W	0x0000	26.5.2.68/26-73
0x041A	MBCCFR99–Message buffer cycle counter filter register 99	R/W	—[3]	26.5.2.69/26-75
0x041C	MBFIDR99–Message buffer frame ID register 99	R/W	0x0UUU	26.5.2.70/26-76
0x041E	MBIDXR99–Message buffer index register 99	R/W	0x00UU	26.5.2.71/26-76
0x0420	MBCCSR100–Message buffer configuration, control, status register 100	R/W	0x0000	26.5.2.68/26-73
0x0422	MBCCFR100–Message buffer cycle counter filter register 100	R/W	—[3]	26.5.2.69/26-75
0x0424	MBFIDR100–Message buffer frame ID register 100	R/W	0x0UUU	26.5.2.70/26-76
0x0426	MBIDXR100–Message buffer index register 100	R/W	0x00UU	26.5.2.71/26-76
0x0428	MBCCSR101–Message buffer configuration, control, status register 101	R/W	0x0000	26.5.2.68/26-73
0x042A	MBCCFR101–Message buffer cycle counter filter register 101	R/W	—[3]	26.5.2.69/26-75
0x042C	MBFIDR101–Message buffer frame ID register 101	R/W	0x0UUU	26.5.2.70/26-76
0x042E	MBIDXR101–Message buffer index register 101	R/W	0x00UU	26.5.2.71/26-76
0x0430	MBCCSR102–Message buffer configuration, control, status register 102	R/W	0x0000	26.5.2.68/26-73
0x0432	MBCCFR102–Message buffer cycle counter filter register 102	R/W	—[3]	26.5.2.69/26-75
0x0434	MBFIDR102–Message buffer frame ID register 102	R/W	0x0UUU	26.5.2.70/26-76
0x0436	MBIDXR102–Message buffer index register 102	R/W	0x00UU	26.5.2.71/26-76
0x0438	MBCCSR103–Message buffer configuration, control, status register 103	R/W	0x0000	26.5.2.68/26-73
0x043A	MBCCFR103–Message buffer cycle counter filter register 103	R/W	—[3]	26.5.2.69/26-75
0x043C	MBFIDR103–Message buffer frame ID register 103	R/W	0x0UUU	26.5.2.70/26-76
0x043E	MBIDXR103–Message buffer index register 103	R/W	0x00UU	26.5.2.71/26-76
0x0440	MBCCSR104–Message buffer configuration, control, status register 104	R/W	0x0000	26.5.2.68/26-73
0x0442	MBCCFR104–Message buffer cycle counter filter register 104	R/W	—[3]	26.5.2.69/26-75
0x0444	MBFIDR104–Message buffer frame ID register 104	R/W	0x0UUU	26.5.2.70/26-76
0x0446	MBIDXR104–Message buffer index register 104	R/W	0x00UU	26.5.2.71/26-76
0x0448	MBCCSR105–Message buffer configuration, control, status register 105	R/W	0x0000	26.5.2.68/26-73
0x044A	MBCCFR105–Message buffer cycle counter filter register 105	R/W	—[3]	26.5.2.69/26-75
0x044C	MBFIDR105–Message buffer frame ID register 105	R/W	0x0UUU	26.5.2.70/26-76
0x044E	MBIDXR105–Message buffer index register 105	R/W	0x00UU	26.5.2.71/26-76

Table A-4. PXN20 Detailed Register Map (continued)

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
0x0450	MBCCSR106–Message buffer configuration, control, status register 106	R/W	0x0000	26.5.2.68/26-73
0x0452	MBCCFR106–Message buffer cycle counter filter register 106	R/W	—[3]	26.5.2.69/26-75
0x0454	MBFIDR106–Message buffer frame ID register 106	R/W	0x0UUU	26.5.2.70/26-76
0x0456	MBIDXR106–Message buffer index register 106	R/W	0x00UU	26.5.2.71/26-76
0x0458	MBCCSR107–Message buffer configuration, control, status register 107	R/W	0x0000	26.5.2.68/26-73
0x045A	MBCCFR107–Message buffer cycle counter filter register 107	R/W	—[3]	26.5.2.69/26-75
0x045C	MBFIDR107–Message buffer frame ID register 107	R/W	0x0UUU	26.5.2.70/26-76
0x045E	MBIDXR107–Message buffer index register 107	R/W	0x00UU	26.5.2.71/26-76
0x0460	MBCCSR108–Message buffer configuration, control, status register 108	R/W	0x0000	26.5.2.68/26-73
0x0462	MBCCFR108–Message buffer cycle counter filter register 108	R/W	—[3]	26.5.2.69/26-75
0x0464	MBFIDR108–Message buffer frame ID register 108	R/W	0x0UUU	26.5.2.70/26-76
0x0466	MBIDXR108–Message buffer index register 108	R/W	0x00UU	26.5.2.71/26-76
0x0468	MBCCSR109–Message buffer configuration, control, status register 109	R/W	0x0000	26.5.2.68/26-73
0x046A	MBCCFR109–Message buffer cycle counter filter register 109	R/W	—[3]	26.5.2.69/26-75
0x046C	MBFIDR109–Message buffer frame ID register 109	R/W	0x0UUU	26.5.2.70/26-76
0x046E	MBIDXR109–Message buffer index register 109	R/W	0x00UU	26.5.2.71/26-76
0x0470	MBCCSR110–Message buffer configuration, control, status register 110	R/W	0x0000	26.5.2.68/26-73
0x0472	MBCCFR110–Message buffer cycle counter filter register 110	R/W	—[3]	26.5.2.69/26-75
0x0474	MBFIDR110–Message buffer frame ID register 110	R/W	0x0UUU	26.5.2.70/26-76
0x0476	MBIDXR110–Message buffer index register 110	R/W	0x00UU	26.5.2.71/26-76
0x0478	MBCCSR111–Message buffer configuration, control, status register 111	R/W	0x0000	26.5.2.68/26-73
0x047A	MBCCFR111–Message buffer cycle counter filter register 111	R/W	—[3]	26.5.2.69/26-75
0x047C	MBFIDR111–Message buffer frame ID register 111	R/W	0x0UUU	26.5.2.70/26-76
0x047E	MBIDXR111–Message buffer index register 111	R/W	0x00UU	26.5.2.71/26-76
0x0480	MBCCSR112–Message buffer configuration, control, status register 112	R/W	0x0000	26.5.2.68/26-73
0x0482	MBCCFR112–Message buffer cycle counter filter register 112	R/W	—[3]	26.5.2.69/26-75
0x0484	MBFIDR112–Message buffer frame ID register 112	R/W	0x0UUU	26.5.2.70/26-76
0x0486	MBIDXR112–Message buffer index register 112	R/W	0x00UU	26.5.2.71/26-76

Table A-4. PXN20 Detailed Register Map (continued)

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
0x0488	MBCCSR113–Message buffer configuration, control, status register 113	R/W	0x0000	26.5.2.68/26-73
0x048A	MBCCFR13–Message buffer cycle counter filter register 113	R/W	—[3]	26.5.2.69/26-75
0x048C	MBFIDR113–Message buffer frame ID register 113	R/W	0x0UUU	26.5.2.70/26-76
0x048E	MBIDXR113–Message buffer index register 113	R/W	0x00UU	26.5.2.71/26-76
0x0490	MBCCSR114–Message buffer configuration, control, status register 114	R/W	0x0000	26.5.2.68/26-73
0x0492	MBCCFR14–Message buffer cycle counter filter register 114	R/W	—[3]	26.5.2.69/26-75
0x0494	MBFIDR114–Message buffer frame ID register 114	R/W	0x0UUU	26.5.2.70/26-76
0x0496	MBIDXR114–Message buffer index register 114	R/W	0x00UU	26.5.2.71/26-76
0x0498	MBCCSR115–Message buffer configuration, control, status register 115	R/W	0x0000	26.5.2.68/26-73
0x049A	MBCCFR115–Message buffer cycle counter filter register 115	R/W	—[3]	26.5.2.69/26-75
0x049C	MBFIDR115–Message buffer frame ID register 115	R/W	0x0UUU	26.5.2.70/26-76
0x049E	MBIDXR115–Message buffer index register 115	R/W	0x00UU	26.5.2.71/26-76
0x04A0	MBCCSR116–Message buffer configuration, control, status register 116	R/W	0x0000	26.5.2.68/26-73
0x04A2	MBCCFR116–Message buffer cycle counter filter register 116	R/W	—[3]	26.5.2.69/26-75
0x04A4	MBFIDR116–Message buffer frame ID register 161	R/W	0x0UUU	26.5.2.70/26-76
0x04A6	MBIDXR116–Message buffer index register 116	R/W	0x00UU	26.5.2.71/26-76
0x04A8	MBCCSR117–Message buffer configuration, control, status register 117	R/W	0x0000	26.5.2.68/26-73
0x04AA	MBCCFR117–Message buffer cycle counter filter register 117	R/W	—[3]	26.5.2.69/26-75
0x04AC	MBFIDR117–Message buffer frame ID register 117	R/W	0x0UUU	26.5.2.70/26-76
0x04AE	MBIDXR117–Message buffer index register 117	R/W	0x00UU	26.5.2.71/26-76
0x04B0	MBCCSR118–Message buffer configuration, control, status register 118	R/W	0x0000	26.5.2.68/26-73
0x04B2	MBCCFR118–Message buffer cycle counter filter register 118	R/W	—[3]	26.5.2.69/26-75
0x04B4	MBFIDR118–Message buffer frame ID register 118	R/W	0x0UUU	26.5.2.70/26-76
0x04B6	MBIDXR118–Message buffer index register 118	R/W	0x00UU	26.5.2.71/26-76
0x04B8	MBCCSR119–Message buffer configuration, control, status register 119	R/W	0x0000	26.5.2.68/26-73
0x04BA	MBCCFR119–Message buffer cycle counter filter register 119	R/W	—[3]	26.5.2.69/26-75
0x04BC	MBFIDR119–Message buffer frame ID register 119	R/W	0x0UUU	26.5.2.70/26-76
0x04BE	MBIDXR119–Message buffer index register 119	R/W	0x00UU	26.5.2.71/26-76

Table A-4. PXN20 Detailed Register Map (continued)

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
0x04C0	MBCCSR120–Message buffer configuration, control, status register 120	R/W	0x0000	26.5.2.68/26-73
0x04C2	MBCCFR120–Message buffer cycle counter filter register 120	R/W	—[3]	26.5.2.69/26-75
0x04C4	MBFIDR120–Message buffer frame ID register 120	R/W	0x0UUU	26.5.2.70/26-76
0x04C6	MBIDXR120–Message buffer index register 120	R/W	0x00UU	26.5.2.71/26-76
0x04C8	MBCCSR121–Message buffer configuration, control, status register 121	R/W	0x0000	26.5.2.68/26-73
0x04CA	MBCCFR121–Message buffer cycle counter filter register 121	R/W	—[3]	26.5.2.69/26-75
0x04CC	MBFIDR121–Message buffer frame ID register 121	R/W	0x0UUU	26.5.2.70/26-76
0x04CE	MBIDXR121–Message buffer index register 121	R/W	0x00UU	26.5.2.71/26-76
0x04D0	MBCCSR122–Message buffer configuration, control, status register 122	R/W	0x0000	26.5.2.68/26-73
0x04D2	MBCCFR122–Message buffer cycle counter filter register 122	R/W	—[3]	26.5.2.69/26-75
0x04D4	MBFIDR122–Message buffer frame ID register122	R/W	0x0UUU	26.5.2.70/26-76
0x04D6	MBIDXR122–Message buffer index register 122	R/W	0x00UU	26.5.2.71/26-76
0x04D8	MBCCSR123–Message buffer configuration, control, status register 123	R/W	0x0000	26.5.2.68/26-73
0x04DA	MBCCFR123–Message buffer cycle counter filter register 123	R/W	—[3]	26.5.2.69/26-75
0x04DC	MBFIDR123–Message buffer frame ID register 123	R/W	0x0UUU	26.5.2.70/26-76
0x04DE	MBIDXR123–Message buffer index register 123	R/W	0x00UU	26.5.2.71/26-76
0x04E0	MBCCSR124–Message buffer configuration, control, status register 124	R/W	0x0000	26.5.2.68/26-73
0x04E2	MBCCFR124–Message buffer cycle counter filter register 124	R/W	—[3]	26.5.2.69/26-75
0x04E4	MBFIDR124–Message buffer frame ID register 124	R/W	0x0UUU	26.5.2.70/26-76
0x04E6	MBIDXR124–Message buffer index register 124	R/W	0x00UU	26.5.2.71/26-76
0x04E8	MBCCSR125–Message buffer configuration, control, status register 125	R/W	0x0000	26.5.2.68/26-73
0x04EA	MBCCFR125–Message buffer cycle counter filter register 125	R/W	—[3]	26.5.2.69/26-75
0x04EC	MBFIDR125–Message buffer frame ID register 125	R/W	0x0UUU	26.5.2.70/26-76
0x04EE	MBIDXR125–Message buffer index register 125	R/W	0x00UU	26.5.2.71/26-76
0x04F0	MBCCSR126–Message buffer configuration, control, status register 126	R/W	0x0000	26.5.2.68/26-73
0x04F2	MBCCFR26–Message buffer cycle counter filter register 126	R/W	—[3]	26.5.2.69/26-75
0x04F4	MBFIDR126–Message buffer frame ID register 126	R/W	0x0UUU	26.5.2.70/26-76
0x04F6	MBIDXR126–Message buffer index register 126	R/W	0x00UU	26.5.2.71/26-76

Table A-4. PXN20 Detailed Register Map (continued)

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
0x04F8	MBCCSR127–Message buffer configuration, control, status register 127	R/W	0x0000	26.5.2.68/26-73
0x04FA	MBCCFR127–Message buffer cycle counter filter register 127	R/W	—[3]	26.5.2.69/26-75
0x04FC	MBFIDR127–Message buffer frame ID register 127	R/W	0x0UUU	26.5.2.70/26-76
0x04FE	MBIDXR127–Message buffer index register 127	R/W	0x00UU	26.5.2.71/26-76
0x0500–0x07FF	Reserved			
0x0800–0x3FFF	FlexRay memory	R/W	—[3]	
0xC3FE_0000–0xDFFF_FFFF	**Reserved**			
Peripherals AIPS_B (AXBS Port S7) 0xFFF0_4000–0xFFFF_FFFF				
0xE000_0000–0xFFF0_3FFF	**Reserved**			
0xFFF0_4000	**.AXBS_BASE Chapter 16, AMBA Crossbar Switch (AXBS)**			
0x0000	XBAR_MPR0—Master priority register, slave port 0	R/W	0x5400_3210	16.2.1.1/16-4
0x0004–0x000F	Reserved			
0x0010	XBAR_SGPCR0—General-purpose control register, slave port 0	R/W	0x0000_0000	16.2.1.2/16-6
0x0004–0x000F	Reserved			
0x0100	XBAR_MPR1—Master priority register, slave port 1	R/W	0x5400_3210	16.2.1.1/16-4
0x0104–0x010F	Reserved			
0x0110	XBAR_SGPCR1—General-purpose control register, slave port 1	R/W	0x0000_0000	16.2.1.2/16-6
0x0114– 0x01FF	Reserved			
0x0200	XBAR_MPR2—Master priority register, slave port 2	R/W	0x5400_3210	16.2.1.1/16-4
0x0204–0x020F	Reserved			
0x0210	XBAR_SGPCR2—General-purpose control register, slave port 2	R/W	0x0000_0000	16.2.1.2/16-6
0x0214– 0x02FF	Reserved			
0x0300	XBAR_MPR3—Master priority register, slave port 3	R/W	0x5400_3210	16.2.1.1/16-4
0x0304–0x030F	Reserved			
0x0310	XBAR_SGPCR3—General-purpose control register, slave port 3	R/W	0x0000_0000	16.2.1.2/16-6
0x0314–0x05FF	Reserved			

Table A-4. PXN20 Detailed Register Map (continued)

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
0x0600	XBAR_MPR6—Master priority register, slave port 6	R/W	0x5400_3210	16.2.1.1/16-4
0x0604–0x060F	Reserved			
0x0610	XBAR_SGPCR6—General-purpose control register, slave port 6	R/W	0x0000_0000	16.2.1.2/16-6
0x0614–0x06FF	Reserved			
0x0700	XBAR_MPR7—Master priority register, slave port 7	R/W	0x5400_3210	16.2.1.1/16-4
0x0704–0x070F	Reserved			
0x0710	XBAR_SGPCR7— General-purpose control register, slave port 7	R/W	0x0000_0000	16.2.1.2/16-6
0x0714–0x0EFF	Reserved			
0x0F00	XBAR_MGPCR7—Master General Purpose Register, Master Port 7	R/W	0x0000_0000	16.2.1.3/16-8
0x0F04–0x_3FFF	Reserved			
0xFFF0_8000 – 0xFFF0_FFFF	**Reserved**			
0xFFF1_0000	**SEMAPHORES** **Chapter 15, Semaphores**			
0x0000	SEMA4_Gate00—Semaphores gate 0	R/W	0x00	15.3.2.1/15-4
0x0001	SEMA4_Gate01—Semaphores gate 1	R/W	0x00	15.3.2.1/15-4
0x0002	SEMA4_Gate02—Semaphores gate 2	R/W	0x00	15.3.2.1/15-4
0x0003	SEMA4_Gate03—Semaphores gate 3	R/W	0x00	15.3.2.1/15-4
0x0004	SEMA4_Gate04—Semaphores gate 4	R/W	0x00	15.3.2.1/15-4
0x0005	SEMA4_Gate05—Semaphores gate 5	R/W	0x00	15.3.2.1/15-4
0x0006	SEMA4_Gate06—Semaphores gate 6	R/W	0x00	15.3.2.1/15-4
0x0007	SEMA4_Gate07—Semaphores gate 7	R/W	0x00	15.3.2.1/15-4
0x0008	SEMA4_Gate08—Semaphores gate 8	R/W	0x00	15.3.2.1/15-4
0x0009	SEMA4_Gate09—Semaphores gate 9	R/W	0x00	15.3.2.1/15-4
0x000A	SEMA4_Gate10—Semaphores gate 10	R/W	0x00	15.3.2.1/15-4
0x000B	SEMA4_Gate11—Semaphores gate 11	R/W	0x00	15.3.2.1/15-4
0x000C	SEMA4_Gate12—Semaphores gate 12	R/W	0x00	15.3.2.1/15-4
0x000D	SEMA4_Gate13—Semaphores gate 13	R/W	0x00	15.3.2.1/15-4
0x000E	SEMA4_Gate14—Semaphores gate 14	R/W	0x00	15.3.2.1/15-4
0x000F	SEMA4_Gate15—Semaphores gate 15	R/W	0x00	15.3.2.1/15-4
0x0010–0x003F	Reserved			

Table A-4. PXN20 Detailed Register Map (continued)

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
00x040	SEMA4_CP0INE—Semaphores CP0 IRQ notification enable	R/W	0x0000	15.3.2.2/15-5
0x0042–0x0047	Reserved			
0x0048	SEMA4_CP1INE—Semaphores CP1 IRQ notification enable	R/W	0x0000	15.3.2.2/15-5
0x004A–0x07F	Reserved			
0x0080	SEMA4_CP0NTF—Semaphores CP0 IRQ notification	R	0x0000	15.3.2.3/15-6
0x008 2–00x087	Reserved			
0x0088	SEMA4_CP1NTF—Semaphores CP1 IRQ notification	R	0x0000	15.3.2.2/15-5
0x008A–0x00FF	Reserved			
0x0100	SEMA4_RSTGT—Semaphores reset gate	R/W	0x0000	15.3.2.4/15-6
0x0102	Reserved			
0x0104	SEMA4_RSTNTF—Semaphores reset IRQ notification	R/W	0x00000	15.3.2.5/15-8
0x0106–0x3FFF	Reserved			
0xFFF1_4000	**MPU** **Chapter 18, Memory Protection Unit (MPU)**			
0x0000	MPU_CESR—MPU control/error status register	R/W	0x0080_3200	18.3.2.1/18-5
0x0004–0x000F	Reserved			
0x0010	MPU_EAR0—MPU error address register, MPU port 0	R	—[3]	18.3.2.2/18-6
0x0014	MPU_EDR0—MPU error detail register, MPU port 0	R	—[3]	18.3.2.3/18-7
0x0018	MPU_EAR1—MPU error address register, MPU port 1	R	—[3]	18.3.2.2/18-6
0x001C	MPU_EDR1—MPU error detail register, MPU port 1	R	—[3]	18.3.2.3/18-7
0x0020	MPU_EAR2—MPU error address register, MPU port 2	R	—[3]	18.3.2.2/18-6
0x0024	MPU_EDR2—MPU error detail register, MPU port 2	R	—[3]	18.3.2.3/18-7
0x0028	MPU_EAR3— MPU error address register, MPU port 3	R	—[3]	18.3.2.3/18-7
0x002C	MPU_EDR3—MPU error detail register, MPU port 3	R	—[3]	18.3.2.3/18-7
0x0030–0x03FF	Reserved			
0x0400	MPU_RGD0—MPU region descriptor 0	R/W	—[3]	18.3.2.4/18-8
0x0410	MPU_RGD1—MPU region descriptor 1	R/W	—[3]	18.3.2.4/18-8
0x0420	MPU_RGD2—MPU region descriptor 2	R/W	—[3]	18.3.2.4/18-8
0x0430	MPU_RGD3—MPU region descriptor 3	R/W	—[3]	18.3.2.4/18-8
0x0440	MPU_RGD4—MPU region descriptor 4	R/W	—[3]	18.3.2.4/18-8
0x0450	MPU_RGD5—MPU region descriptor 5	R/W	—[3]	18.3.2.4/18-8
0x0460	MPU_RGD6—MPU region descriptor 6	R/W	—[3]	18.3.2.4/18-8

Table A-4. PXN20 Detailed Register Map (continued)

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
0x0470	MPU_RGD7—MPU region descriptor 7	R/W	—[3]	18.3.2.4/18-8
0x0480	MPU_RGD8—MPU region descriptor 8	R/W	—[3]	18.3.2.4/18-8
0x0490	MPU_RGD9—MPU region descriptor 9	R/W	—[3]	18.3.2.4/18-8
0x04A0	MPU_RGD10—MPU region descriptor 10	R/W	—[3]	18.3.2.4/18-8
0x04B0	MPU_RGD11—MPU region descriptor 11	R/W	—[3]	18.3.2.4/18-8
0x04C0	MPU_RGD12—MPU region descriptor 12	R/W	—[3]	18.3.2.4/18-8
0x04D0	MPU_RGD13—MPU region descriptor 13	R/W	—[3]	18.3.2.4/18-8
0x04E0	MPU_RGD14—MPU region descriptor 14	R/W	—[3]	18.3.2.4/18-8
0x04F0	MPU_RGD15—MPU region descriptor 15	R/W	—[3]	18.3.2.4/18-8
0x00500–0x07FF	Reserved			
0x0800	MPU_RGDAAC0—MPU RGD alternate access control 0	R/W	—[3]	18.3.2.5/18-13
0x0804	MPU_RGDAAC1—MPU RGD alternate access control 1	R/W	—[3]	18.3.2.5/18-13
0x0808	MPU_RGDAAC2—MPU RGD alternate access control 2	R/W	—[3]	18.3.2.5/18-13
0x080C	MPU_RGDAAC3—MPU RGD alternate access control 3	R/W	—[3]	18.3.2.5/18-13
0x0810	MPU_RGDAAC4—MPU RGD alternate access control 4	R/W	—[3]	18.3.2.5/18-13
0x0814	MPU_RGDAAC5—MPU RGD alternate access control 5	R/W	—[3]	18.3.2.5/18-13
0x0818	MPU_RGDAAC6—MPU RGD alternate access control 6	R/W	—[3]	18.3.2.5/18-13
0x081C	MPU_RGDAAC7—MPU RGD alternate access control 7	R/W	—[3]	18.3.2.5/18-13
0x0820	MPU_RGDAAC8—MPU RGD alternate access control 8	R/W	—[3]	18.3.2.5/18-13
0x0824	MPU_RGDAAC9—MPU RGD alternate access control 9	R/W	—[3]	18.3.2.5/18-13
0x0828	MPU_RGDAAC10—MPU RGD alternate access control 10	R/W	—[3]	18.3.2.5/18-13
0x082C	MPU_RGDAAC11—MPU RGD alternate access control 11	R/W	—[3]	18.3.2.5/18-13
0x0830	MPU_RGDAAC12—MPU RGD alternate access control 12	R/W	—[3]	18.3.2.5/18-13
0x0834	MPU_RGDAAC13—MPU RGD alternate access control 13	R/W	—[3]	18.3.2.5/18-13
0x0838	MPU_RGDAAC14—MPU RGD alternate access control 14	R/W	—[3]	18.3.2.5/18-13
0x083C	MPU_RGDAAC15—MPU RGD alternate access control 15	R/W	—[3]	18.3.2.5/18-13
0x0840–0x3FFF	Reserved			
0xFFF1_8000–0xFFF3_7FFF	**Reserved**			
0xFFF3_8000	**SWT** **Chapter 20, Software Watchdog Timer (SWT)**			
0x0000	SWT_CR – SWT control register	R/W	0xFF00_011B	20.3.2.1/20-2

Table A-4. PXN20 Detailed Register Map (continued)

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
0x0004	SWT_IR – SWT interrupt register	R/W	0x0000_0000	20.3.2.2/20-4
0x0008	SWT_TO – SWT time-out register	R/W	0x0002_7FFF	20.3.2.3/20-4
0x000C	SWT_WN – SWT window register	R/W	0x0000_0000	20.3.2.4/20-5
0x0010	SWT_SR – SWT service register	R/W	0x0000_0000	20.3.2.5/20-6
0x0014	SWT_CO – SWT counter output register	R	0x0000_0000	20.3.2.6/20-6
0x0018	SWT_SK – SWT service key register	R/W	0x0000_0000	20.3.2.7/20-7
0x001C– 0x3FFF	Reserved			
0xFFF3_C000	**STM** **Chapter 21, System Timer Module (STM)**			
0x0000	STM_CR – STM control register	R/W	0x0000_0000	21.3.2.1/21-3
0x0004	STM_CNT – STM counter value	R/W	0x0000_0000	21.3.2.2/21-3
0x0008–0x000F	Reserved			
0x0010	STM_CCR0 – STM channel 0 control register	R/W	0x0000_0000	21.3.2.3/21-4
0x0014	STM_CIR0 – STM channel 0 interrupt register	R/W	0x0000_0000	21.3.2.4/21-4
0x0018	STM_CMP0 – STM channel 0 compare register	R/W	0x0000_0000	21.3.2.5/21-5
0x001C	Reserved			
0x0020	STM_CCR1 – STM channel 1 control register	R/W	0x0000_0000	21.3.2.3/21-4
0x0024	STM_CIR1 – STM channel 1 interrupt register	R/W	0x0000_0000	21.3.2.4/21-4
0x0028	STM_CMP1 – STM channel 1 compare register	R/W	0x0000_0000	21.3.2.5/21-5
0x002C	Reserved			
0x0030	STM_CCR2 – STM channel 2 control register	R/W	0x0000_0000	21.3.2.3/21-4
0x0034	STM_CIR2 – STM channel 2 interrupt register	R/W	0x0000_0000	21.3.2.4/21-4
0x0038	STM_CMP2 – STM channel 2 compare register	R/W	0x0000_0000	21.3.2.5/21-5
0x003C	Reserved			
0x0040	STM_CCR3 – STM channel 3 control register	R/W	0x0000_0000	21.3.2.3/21-4
0x0044	STM_CIR3 – STM channel 3 interrupt register	R/W	0x0000_0000	21.3.2.4/21-4
0x0048	STM_CMP3 – STM channel 3 compare register	R/W	0x0000_0000	21.3.2.5/21-5
0x004C–0x3FFF	Reserved			
0xFFF4_0000	**ECSM** **Chapter 19, Error Correction Status Module (ECSM)**			
0x0000–0x0023	Reserved			
0x0024	FBOMCR—FEC burst optimization master control register	R/W	0x0000_0000	19.2.2.1/19-3

Table A-4. PXN20 Detailed Register Map (continued)

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
0x0028–0x0042	Reserved			
0x0043	ECR—ECC Configuration	R/W	0x00	19.2.2.2/19-5
0x0047	ESR—ECC status	R/W	0x00	19.2.2.3/19-6
0x004A	EEGR—ECC error generation	R/W	0x0000	19.2.2.4/19-7
0x0050	PFEAR—PFlash ECC address	R	—[3]	19.2.2.5/19-9
0x0056	PFEMR—PFlash ECC master	R	0x0U	19.2.2.6/19-10
0x0057	PFEAT—PFlash ECC attributes register	R	—[3]	19.2.2.7/19-10
0x0058	PFEDRH—PFlash ECC data register high	R	—[3]	19.2.2.8/19-11
0x005C	PFEDRL—PFlash ECC data register low	R	—[3]	19.2.2.8/19-11
0x0060	PREAR—PRAM ECC address	R	—[3]	19.2.2.9/19-12
0x0065	PRESR—PRAM ECC syndrome register	R	—[3]	19.2.2.10/19-13
0x0066	PREMR—PRAM ECC master	R	0x0U	19.2.2.11/19-14
0x0067	PREAT—PRAM ECC attributes	R	—[3]	19.2.2.12/19-15
0x0068	PREDRH—PRAM ECC data register high	R	—[3]	19.2.2.13/19-16
0x006C	PREDRL—PRAM ECC data register low	R	—[3]	19.2.2.13/19-16
0x0070–0x3FFF	Reserved			
0xFFF4_4000	**eDMA** **Chapter 24, Enhanced Direct Memory Access Controller (eDMA)**			
0x0000	EDMA_CR—eDMA control register	R/W	0x0000_0400	24.3.2.1/24-8
0x0004	EDMA_ESR—eDMA error status register	R	0x0000_0000	24.3.2.2/24-10
0x0008	Reserved			
0x000C	EDMA_ERQRL—eDMA enable request low register (channels 31–00)	R/W	0x0000_0000	24.3.2.3/24-12
0x0010	Reserved			
0x0014	EDMA_EEIRL—eDMA enable error interrupt low register (channels 31–00)	R/W	0x0000_0000	24.3.2.4/24-13
0x0018	EDMA_SERQR—eDMA set enable request register	W	0x00	24.3.2.5/24-14
0x0019	EDMA_CERQR—eDMA clear enable request register	W	0x00	24.3.2.6/24-15
0x001A	EDMA_SEEIR—eDMA set enable error interrupt register	W	0x00	24.3.2.7/24-15
0x001B	EDMA_CEEIR—eDMA clear enable error interrupt register	W	0x00	24.3.2.8/24-16
0x001C	EDMA_CIRQR—eDMA clear interrupt request register	W	0x00	24.3.2.9/24-17
0x001D	EDMA_CER—eDMA clear error register	W	0x00	24.3.2.10/24-18
0x001E	EDMA_SSBR—eDMA set start bit register	W	0x00	24.3.2.11/24-18

Table A-4. PXN20 Detailed Register Map (continued)

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
0x001F	EDMA_CDSBR—eDMA clear done status bit register	W	0x00	24.3.2.12/24-19
0x0020	Reserved			
0x0024	EDMA_IRQRL—eDMA interrupt request low register (channels 31–00)	R/W	0x0000_0000	24.3.2.13/24-19
0x0028	Reserved			
0x002C	EDMA_ERL—eDMA error low register (channels 31–00)	R/W	0x0000_0000	24.3.2.14/24-20
0x0030	Reserved			
0x0034	EDMA_HRSL—eDMA hardware request status register (channels 31–00)	R/W	0x0000_0000	24.3.2.15/24-21
0x0038–0x00FF	Reserved			
0x0100	EDMA_CPR0—eDMA channel 0 priority register	R/W	0x00	
0x0101	EDMA_CPR1—eDMA channel 1 priority register	R/W	0x01	24.3.2.16/24-22
0x0102	EDMA_CPR2—eDMA channel 2 priority register	R/W	0x02	24.3.2.16/24-22
0x0103	EDMA_CPR3—eDMA channel 3 priority register	R/W	0x03	24.3.2.16/24-22
0x0104	EDMA_CPR4—eDMA channel 4 priority register	R/W	0x04	24.3.2.16/24-22
0x0105	EDMA_CPR5—eDMA channel 5 priority register	R/W	0x05	24.3.2.16/24-22
0x0106	EDMA_CPR6—eDMA channel 6 priority register	R/W	0x06	24.3.2.16/24-22
0x0107	EDMA_CPR7—eDMA channel 7 priority register	R/W	0x07	24.3.2.16/24-22
0x0108	EDMA_CPR8—eDMA channel 8 priority register	R/W	0x08	24.3.2.16/24-22
0x0109	EDMA_CPR9—eDMA channel 9 priority register	R/W	0x09	24.3.2.16/24-22
0x010A	EDMA_CPR10—eDMA channel 10 priority register	R/W	0x0A	24.3.2.16/24-22
0x010B	EDMA_CPR11—eDMA channel 11 priority register	R/W	0x0B	24.3.2.16/24-22
0x010C	EDMA_CPR12—eDMA channel 12 priority register	R/W	0x0C	24.3.2.16/24-22
0x010D	EDMA_CPR13—eDMA channel 13 priority register	R/W	0x0D	24.3.2.16/24-22
0x010E	EDMA_CPR14—eDMA channel 14 priority register	R/W	0x0E	24.3.2.16/24-22
0x010F	EDMA_CPR15—eDMA channel 15 priority register	R/W	0x0F	24.3.2.16/24-22
0x0110	EDMA_CPR16—eDMA channel 16 priority register	R/W	0x10	24.3.2.16/24-22
0x0111	EDMA_CPR17—eDMA channel 17 priority register	R/W	0x11	24.3.2.16/24-22
0x0112	EDMA_CPR18—eDMA channel 18 priority register	R/W	0x12	24.3.2.16/24-22
0x0113	EDMA_CPR19—eDMA channel 19 priority register	R/W	0x13	24.3.2.16/24-22
0x0114	EDMA_CPR20—eDMA channel 20 priority register	R/W	0x14	24.3.2.16/24-22
0x0115	EDMA_CPR21—eDMA channel 21 priority register	R/W	0x15	24.3.2.16/24-22
0x0116	EDMA_CPR22—eDMA channel 22 priority register	R/W	0x16	24.3.2.16/24-22

Table A-4. PXN20 Detailed Register Map (continued)

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
0x0117	EDMA_CPR23—eDMA channel 23 priority register	R/W	0x17	24.3.2.16/24-22
0x0118	EDMA_CPR24—eDMA channel 24 priority register	R/W	0x18	24.3.2.16/24-22
0x0119	EDMA_CPR25—eDMA channel 25 priority register	R/W	0x19	24.3.2.16/24-22
0x011A	EDMA_CPR26—eDMA channel 26 priority register	R/W	0x1A	24.3.2.16/24-22
0x011B	EDMA_CPR27—eDMA channel 27 priority register	R/W	0x1B	24.3.2.16/24-22
0x011C	EDMA_CPR28—eDMA channel 28 priority register	R/W	0x1C	24.3.2.16/24-22
0x011D	EDMA_CPR29—eDMA channel 29 priority register	R/W	0x1D	24.3.2.16/24-22
0x011E	EDMA_CPR30—eDMA channel 30 priority register	R/W	0x1E	24.3.2.16/24-22
0x011F	EDMA_CPR31—eDMA channel 31 priority register	R/W	0x1F	24.3.2.16/24-22
0x0120–0x0FFF	Reserved			
0x1000	TCD00—eDMA transfer control descriptor 00	R/W	—[1]	
0x1020	TCD01—eDMA transfer control descriptor 01	R/W	—[1]	24.3.2.17/24-23
0x1040	TCD02—eDMA transfer control descriptor 02	R/W	—[1]	24.3.2.17/24-23
0x1060	TCD03—eDMA transfer control descriptor 03	R/W	—[1]	24.3.2.17/24-23
0x1080	TCD04—eDMA transfer control descriptor 04	R/W	—[1]	24.3.2.17/24-23
0x10A0	TCD05—eDMA transfer control descriptor 05	R/W	—[1]	24.3.2.17/24-23
0x10C0	TCD06—eDMA transfer control descriptor 06	R/W	—[1]	24.3.2.17/24-23
0x10E0	TCD07—eDMA transfer control descriptor 07	R/W	—[1]	24.3.2.17/24-23
0x1100	TCD08—eDMA transfer control descriptor 08	R/W	—[1]	24.3.2.17/24-23
0x1120	TCD09—eDMA transfer control descriptor 09	R/W	—[1]	24.3.2.17/24-23
0x1140	TCD10—eDMA transfer control descriptor 10	R/W	—[1]	24.3.2.17/24-23
0x1160	TCD11—eDMA transfer control descriptor 11	R/W	—[1]	24.3.2.17/24-23
0x1180	TCD12—eDMA transfer control descriptor 12	R/W	—[1]	24.3.2.17/24-23
0x11A0	TCD13—eDMA transfer control descriptor 13	R/W	—[1]	24.3.2.17/24-23
0x11C0	TCD14—eDMA transfer control descriptor 14	R/W	—[1]	24.3.2.17/24-23
0x11E0	TCD15—eDMA transfer control descriptor 15	R/W	—[1]	24.3.2.17/24-23
0x1200	TCD16—eDMA transfer control descriptor 16	R/W	—[1]	24.3.2.17/24-23
0x1220	TCD17—eDMA transfer control descriptor 17	R/W	—[1]	24.3.2.17/24-23
0x1240	TCD18—eDMA transfer control descriptor 18	R/W	—[1]	24.3.2.17/24-23
0x1260	TCD19—eDMA transfer control descriptor 19	R/W	—[1]	24.3.2.17/24-23
0x1280	TCD20—eDMA transfer control descriptor 20	R/W	—[1]	24.3.2.17/24-23
0x12A0	TCD21—eDMA transfer control descriptor 21	R/W	—[1]	24.3.2.17/24-23

Table A-4. PXN20 Detailed Register Map (continued)

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
0x12C0	TCD22—eDMA transfer control descriptor 22	R/W	—[1]	24.3.2.17/24-23
0x12E0	TCD23—eDMA transfer control descriptor 23	R/W	—[1]	24.3.2.17/24-23
0x1300	TCD24—eDMA transfer control descriptor 24	R/W	—[1]	24.3.2.17/24-23
0x1320	TCD25—eDMA transfer control descriptor 25	R/W	—[1]	24.3.2.17/24-23
0x1340	TCD26—eDMA transfer control descriptor 26	R/W	—[1]	24.3.2.17/24-23
0x1360	TCD27—eDMA transfer control descriptor 27	R/W	—[1]	24.3.2.17/24-23
0x1380	TCD28—eDMA transfer control descriptor 28	R/W	—[1]	24.3.2.17/24-23
0x13A0	TCD29—eDMA transfer control descriptor 29	R/W	—[1]	24.3.2.17/24-23
0x13C0	TCD30—eDMA transfer control descriptor 30	R/W	—[1]	24.3.2.17/24-23
0x13E0	TCD31—eDMA transfer control descriptor 31	R/W	—[1]	24.3.2.17/24-23
0x1400–0x3FFF	Reserved			
0xFFF4_8000	**INTC** **Chapter 10, Interrupts and Interrupt Controller (INTC)**			
0x0000	INTC_MCR—INTC module configuration register	R/W	0x0000_0000	10.3.2.1/10-9
0x0004	Reserved			
0x0008	INTC_CPR_PRC0—INTC current priority register for processor 0 (Z6)	R/W	0x0000_000F	10.3.2.2/10-10
0x00C	INTC_CPR_PRC1—INTC current priority register for processor 1 (Z0)	R/W	0x0000_000F	10.3.2.3/10-12
0x0010	INTC_IACKR_PRC0—INTC interrupt acknowledge register for processor 0 (Z6)	R/W	0x0000_0000	10.3.2.4/10-12
0x0014	INTC_IACKR_PRC1—INTC interrupt acknowledge register for processor 1 (Z0)	R/W	0x0000_0000	10.3.2.5/10-14
0x0018	INTC_EOIR_PRC0—INTC end of interrupt register for processor 0 (Z6)	W	0x0000_0000	10.3.2.6/10-14
0x001C	INTC_EOIR_PRC1—INTC end of interrupt register for processor 1 (Z0)	W	0x0000_0000	10.3.2.7/10-15
0x0020	INTC_SSCIR0_3—INTC software set/clear interrupt register 0–3	R/W	0x0000_0000	10.3.2.8/10-15
0x0024	INTC_SSCIR4_7—INTC software set/clear interrupt register 4–7	R/W	0x0000_0000	10.3.2.8/10-15
0x0028–0x003F	Reserved			
0x0040	INTC_PSR0_3—INTC priority select register 0–3	R/W	0x0000_0000	10.3.2.9/10-16
0x0044	INTC_PSR4_7—INTC priority select register 4–7	R/W	0x0000_0000	10.3.2.9/10-16
0x0048	INTC_PSR8_11—INTC priority select register 8–11	R/W	0x0000_0000	10.3.2.9/10-16

Table A-4. PXN20 Detailed Register Map (continued)

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
0x004C	INTC_PSR12_15—INTC priority select register 12–15	R/W	0x0000_0000	10.3.2.9/10-16
0x0050	INTC_PSR16_19—INTC priority select register 16–19	R/W	0x0000_0000	10.3.2.9/10-16
0x0054	INTC_PSR20_23—INTC priority select register 20–23	R/W	0x0000_0000	10.3.2.9/10-16
0x0058	INTC_PSR24_27—INTC priority select register 24–27	R/W	0x0000_0000	10.3.2.9/10-16
0x005C	INTC_PSR28_31—INTC priority select register 24–27	R/W	0x0000_0000	10.3.2.9/10-16
0x0060	INTC_PSR32_35—INTC priority select register 32–35	R/W	0x0000_0000	10.3.2.9/10-16
0x0064	INTC_PSR36_39—INTC priority select register 36–39	R/W	0x0000_0000	10.3.2.9/10-16
0x0068	INTC_PSR40_43—INTC priority select register 40–43	R/W	0x0000_0000	10.3.2.9/10-16
0x006C	INTC_PSR44_47—INTC priority select register 44–47	R/W	0x0000_0000	10.3.2.9/10-16
0x0070	INTC_PSR48_51—INTC priority select register 48–51	R/W	0x0000_0000	10.3.2.9/10-16
0x0074	INTC_PSR52_55—INTC priority select register 52–55	R/W	0x0000_0000	10.3.2.9/10-16
0x0078	INTC_PSR56_59—INTC priority select register 56–59	R/W	0x0000_0000	10.3.2.9/10-16
0x007C	INTC_PSR60_63—INTC priority select register 60–63	R/W	0x0000_0000	10.3.2.9/10-16
0x0080	INTC_PSR64_67—INTC priority select register 64–67	R/W	0x0000_0000	10.3.2.9/10-16
0x0084	INTC_PSR68_71—INTC priority select register 68–71	R/W	0x0000_0000	10.3.2.9/10-16
0x0088	INTC_PSR72_75—INTC priority select register 72–75	R/W	0x0000_0000	10.3.2.9/10-16
0x008C	INTC_PSR76_79—INTC priority select register 76–79	R/W	0x0000_0000	10.3.2.9/10-16
0x0090	INTC_PSR80_83—INTC priority select register 80–83	R/W	0x0000_0000	10.3.2.9/10-16
0x0094	INTC_PSR84_87—INTC priority select register 84–87	R/W	0x0000_0000	10.3.2.9/10-16
0x0098	INTC_PSR88_91—INTC priority select register 88–91	R/W	0x0000_0000	10.3.2.9/10-16
0x009C	INTC_PSR92_95—INTC priority select register 92–95	R/W	0x0000_0000	10.3.2.9/10-16
0x00A0	INTC_PSR96_99—INTC priority select register 96–99	R/W	0x0000_0000	10.3.2.9/10-16
0x00A4	INTC_PSR100_103—INTC priority select register 100–103	R/W	0x0000_0000	10.3.2.9/10-16
0x00A8	INTC_PSR104_107—INTC priority select register 104–107	R/W	0x0000_0000	10.3.2.9/10-16
0x00AC	INTC_PSR108_111—INTC priority select register 108–111	R/W	0x0000_0000	10.3.2.9/10-16
0x00B0	INTC_PSR112_115—INTC priority select register 112–115	R/W	0x0000_0000	10.3.2.9/10-16
0x00B4	INTC_PSR116_119—INTC priority select register 116–119	R/W	0x0000_0000	10.3.2.9/10-16
0x00B8	INTC_PSR120_123—INTC priority select register 120–123	R/W	0x0000_0000	10.3.2.9/10-16
0x00BC	INTC_PSR124_127—INTC priority select register 124–127	R/W	0x0000_0000	10.3.2.9/10-16
0x00C0	INTC_PSR128_131—INTC priority select register 128–131	R/W	0x0000_0000	10.3.2.9/10-16
0x00C4	INTC_PSR132_135—INTC priority select register 132–135	R/W	0x0000_0000	10.3.2.9/10-16
0x00C8	INTC_PSR136_139—INTC priority select register 136–139	R/W	0x0000_0000	10.3.2.9/10-16

Table A-4. PXN20 Detailed Register Map (continued)

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
0x00CC	INTC_PSR140_143—INTC priority select register 140–143	R/W	0x0000_0000	10.3.2.9/10-16
0x00D0	INTC_PSR144_147—INTC priority select register 144–147	R/W	0x0000_0000	10.3.2.9/10-16
0x00D4	INTC_PSR148_151—INTC priority select register 148–151	R/W	0x0000_0000	10.3.2.9/10-16
0x00D8	INTC_PSR152_155—INTC priority select register 152–155	R/W	0x0000_0000	10.3.2.9/10-16
0x00DC	INTC_PSR156_159—INTC priority select register 156–159	R/W	0x0000_0000	10.3.2.9/10-16
0x00E0	INTC_PSR160_163—INTC priority select register 160–163	R/W	0x0000_0000	10.3.2.9/10-16
0x00E4	INTC_PSR164_167—INTC priority select register 164–167	R/W	0x0000_0000	10.3.2.9/10-16
0x00E8	INTC_PSR168_171—INTC priority select register 168–171	R/W	0x0000_0000	10.3.2.9/10-16
0x00EC	INTC_PSR172_175—INTC priority select register 172–175	R/W	0x0000_0000	10.3.2.9/10-16
0x00F0	INTC_PSR176_179—INTC priority select register 176–179	R/W	0x0000_0000	10.3.2.9/10-16
0x00F4	INTC_PSR180_183—INTC priority select register 180–183	R/W	0x0000_0000	10.3.2.9/10-16
0x00F8	INTC_PSR184_187—INTC priority select register 184–187	R/W	0x0000_0000	10.3.2.9/10-16
0x00FC	INTC_PSR188_191—INTC priority select register 188–191	R/W	0x0000_0000	10.3.2.9/10-16
0x0100	INTC_PSR192_195—INTC priority select register 192–195	R/W	0x0000_0000	10.3.2.9/10-16
0x0104	INTC_PSR196_199—INTC priority select register 196–199	R/W	0x0000_0000	10.3.2.9/10-16
0x0108	INTC_PSR200_203—INTC priority select register 200–203	R/W	0x0000_0000	10.3.2.9/10-16
0x010C	INTC_PSR204_207—INTC priority select register 204–207	R/W	0x0000_0000	10.3.2.9/10-16
0x0110	INTC_PSR208_211—INTC priority select register 208–211	R/W	0x0000_0000	10.3.2.9/10-16
0x0114	INTC_PSR212_215—INTC priority select register 212–215	R/W	0x0000_0000	10.3.2.9/10-16
0x0118	INTC_PSR216_219—INTC priority select register 216–219	R/W	0x0000_0000	10.3.2.9/10-16
0x011C	INTC_PSR220_223—INTC priority select register 220–223	R/W	0x0000_0000	10.3.2.9/10-16
0x0120	INTC_PSR224_227—INTC priority select register 224–227	R/W	0x0000_0000	10.3.2.9/10-16
0x0124	INTC_PSR228_231—INTC priority select register 228–231	R/W	0x0000_0000	10.3.2.9/10-16
0x0128	INTC_PSR232_235—INTC priority select register 232–235	R/W	0x0000_0000	10.3.2.9/10-16
0x012C	INTC_PSR236_239—INTC priority select register 236–239	R/W	0x0000_0000	10.3.2.9/10-16
0x0130	INTC_PSR240_243—INTC priority select register 240–243	R/W	0x0000_0000	10.3.2.9/10-16
0x0134	INTC_PSR244_247—INTC priority select register 244–247	R/W	0x0000_0000	10.3.2.9/10-16
0x0138	INTC_PSR248_251—INTC priority select register 248–251	R/W	0x0000_0000	10.3.2.9/10-16
0x013C	INTC_PSR252_255—INTC priority select register 252–255	R/W	0x0000_0000	10.3.2.9/10-16
0x0140	INTC_PSR256_259—INTC priority select register 256–259	R/W	0x0000_0000	10.3.2.9/10-16
0x0144	INTC_PSR260_263—INTC priority select register 260–263	R/W	0x0000_0000	10.3.2.9/10-16
0x0148	INTC_PSR264_267—INTC priority select register 264–267	R/W	0x0000_0000	10.3.2.9/10-16

Table A-4. PXN20 Detailed Register Map (continued)

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
0x014C	INTC_PSR268_271—INTC priority select register 268–271	R/W	0x0000_0000	10.3.2.9/10-16
0x0150	INTC_PSR272_275—INTC priority select register 272–275	R/W	0x0000_0000	10.3.2.9/10-16
0x0154	INTC_PSR276_279—INTC priority select register 276–279	R/W	0x0000_0000	10.3.2.9/10-16
0x0158	INTC_PSR280_283—INTC priority select register 280–283	R/W	0x0000_0000	10.3.2.9/10-16
0x015C	INTC_PSR284_287—INTC priority select register 284–287	R/W	0x0000_0000	10.3.2.9/10-16
0x0160	INTC_PSR288_291—INTC priority select register 288–291	R/W	0x0000_0000	10.3.2.9/10-16
0x0164	INTC_PSR292_295—INTC priority select register 292–295	R/W	0x0000_0000	10.3.2.9/10-16
0x0168	INTC_PSR296_299—INTC priority select register 296–299	R/W	0x0000_0000	10.3.2.9/10-16
0x016C	INTC_PSR300_303—INTC priority select register 300–303	R/W	0x0000_0000	10.3.2.9/10-16
0x0170	INTC_PSR304_307—INTC priority select register 304–307	R/W	0x0000_0000	10.3.2.9/10-16
0x0174	INTC_PSR308_311—INTC priority select register 308–311	R/W	0x0000_0000	10.3.2.9/10-16
0x0178	INTC_PSR312_315—INTC priority select register 312–315	R/W	0x0000_0000	10.3.2.9/10-16
0x017A–0x3FFF	Reserved			
0xFFF4_C000	**FEC** **Chapter 25, Fast Ethernet Controller (FEC)**			
0x0000–0x0003	Reserved			
0x0004	EIR—Interrupt event register	R/W	0x0000_0000	25.3.4.2/25-10
0x0008	EIMR—Interrupt mask register	R/W	0x0000_0000	25.3.4.3/25-12
0x000C–0x000F	Reserved			
0x0010	RDAR—Receive descriptor active register	R/W	0x0000_0000	25.3.4.4/25-12
0x0014	TDAR—Transmit descriptor active register	R/W	0x0000_0000	25.3.4.5/25-13
0x0018–0x0023	Reserved			
0x0024	ECR—Ethernet control register	R/W	0xF000_0000	25.3.4.6/25-14
0x0028–0x003F	Reserved			
0x0040	MMFR—MII management frame register	R/W	—[3]	25.3.4.7/25-15
0x0044	MSCR—MII speed control register	R/W	0x0000_0000	25.3.4.8/25-16
0x0048–0x0063	Reserved			
0x0064	MIBC—MIB control/status register	R/W	0xC000_0000	25.3.4.9/25-18
0x0068–0x0083	Reserved			
0x0084	RCR—Receive control register	R/W	0x05EE_0001	25.3.4.10/25-18
0x0088–0x00C3	Reserved			
0x00C4	TCR—Transmit control register	R/W	0x0000_0000	25.3.4.11/25-20

Table A-4. PXN20 Detailed Register Map (continued)

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
0x00C8–0x00E3	Reserved			
0x00E4	PALR—MAC address low register	R/W	—[3]	25.3.4.12/25-21
0x00E8	PAUR—MAC address upper register + type field	R/W	0xUUUU_8808	25.3.4.13/25-21
0x00EC	OPD—Opcode + pause duration fields	R/W	0x0001_UUUU	25.3.4.14/25-22
0x00F0–0x0117	Reserved			
0x0118	IAUR—Upper 32 bits of individual hash table	R/W	—[3]	25.3.4.15/25-23
0x011C	IALR—Lower 32 bits of individual hash table	R/W	—[3]	25.3.4.16/25-23
0x0120	GAUR—Upper 32 bits of group hash table	R/W	—[3]	25.3.4.17/25-24
0x0124	GALR—Lower 32 bits of group hash table	R/W	—[3]	25.3.4.18/25-25
0x0128–0x0143	Reserved			
0x0144	TFWR—Transmit FIFO watermark	R/W	0x0000_0000	25.3.4.19/25-25
0x0148–0x014B	Reserved			
0x014C	FRBR—FIFO receive bound register	R/W	0x0000_0500	25.3.4.20/25-26
0x0150	FRSR—FIFO receive FIFO start registers	R/W	0x0000_0500	25.3.4.21/25-27
0x0154–0x017F	Reserved			
0x0180	ERDSR—Pointer to receive descriptor ring	R/W	—[3]	25.3.4.22/25-27
0x0184	ETDSR—Transmit buffer descriptor ring start register	R/W	—[3]	25.3.4.23/25-28
0x0188	EMRBR—Receive buffer size register	R/W	—[3]	25.3.4.24/25-29
0x018C–0x3FFF	Reserved			
0xFFF5_0000–0xFFF7_FFFF	**Reserved**			
0xFFF8_0000	**ADC_A** **Chapter 34, Analog-to-Digital Converter (ADC)**			
0x0000	MCR — Main configuration register	R/W[4]	0x0000_0001	34.3.2.1/34-8
0x0004	MSR — Main status register	R/W[1]	0x0000_0001	34.3.2.2/34-10
0x0008–0x000C	Reserved			
0x0010	ISR — Interrupt status register	R/W[1]	0x0000_0000	34.3.2.3/34-12
0x0014	CEOCFR0 — Channel pending register 0	R/W[1]	0x0000_0000	34.3.2.4/34-13
0x0018	CEOCFR1 — Channel pending register 1	R/W[1]	0x0000_0000	34.3.2.5/34-13
0x001C	CEOCFR2 — Channel pending register 2	R/W[1]	0x0000_0000	34.3.2.6/34-14
0x0020	IMR — Interrupt mask register	R/W[1]	0x0000_0000	34.3.2.7/34-15
0x0024	CIMR0 — Channel Interrupt mask register 0	R/W[1]	0x0000_0000	34.3.2.8/34-15

Table A-4. PXN20 Detailed Register Map (continued)

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
0x0028	CIMR1 — Channel Interrupt mask register 1	R/W[1]	0x0000_0000	34.3.2.9/34-16
0x002C	CIMR2 — Channel Interrupt mask register 2	R/W[1]	0x0000_0000	34.3.2.10/34-16
0x0030	WTISR — Watchdog interrupt threshold register	R/W[1]	0x0000_0000	34.3.2.11/34-17
0x0034	WTIMR — Watchdog interrupt threshold mask register	R/W[1]	0x0000_0000	34.3.2.12/34-17
0x0038–0x003C	Reserved			
0x0040	DMAE — DMA enable register	R/W[1]	0x0000_0000	34.3.2.13/34-18
0x0044	DMAR0 — DMA channel select register 0	R/W[1]	0x0000_0000	34.3.2.14/34-19
0x0048	DMAR1 — DMA channel select register 1	R/W[1]	0x0000_0000	34.3.2.15/34-19
0x004C	DMAR2 — DMA channel select register 2	R/W[1]	0x0000_0000	34.3.2.16/34-20
0x0050	TRC0 — Threshold control register 0	R/W[1]	0x0000_0000	34.3.2.17/34-20
0x0054h	TRC1 — Threshold control register 1	R/W[1]	0x0000_0000	34.3.2.17/34-20
0x0058	TRC2 — Threshold control register 2	R/W[1]	0x0000_0000	34.3.2.17/34-20
0x005C	TRC3 — Threshold control register 3	R/W[1]	0x0000_0000	34.3.2.17/34-20
0x0060	THRHLR0 — Threshold register 0	R/W[1]	0x0FFF_0000	34.3.2.18/34-21
0x0064	THRHLR1 — Threshold register 1	R/W[1]	0x0FFF_0000	34.3.2.18/34-21
0x0068	THRHLR2 — Threshold register 2	R/W[1]	0x0FFF_0000	34.3.2.18/34-21
0x006C	THRHLR3 — Threshold register 3	R/W[1]	0x0FFF_0000	34.3.2.18/34-21
0x0070–0x007C	Reserved			
0x0080	PSCR — Presampling control register	R/W[1]	0x0000_0000	34.3.2.19/34-22
0x0084	PSR0 — Presampling register 0	R/W[1]	0x0000_0000	34.3.2.20/34-22
0x0088	PSR1 — Presampling register 1	R/W[1]	0x0000_0000	34.3.2.21/34-23
0x008C	PSR2 — Presampling register 2	R/W[1]	0x0000_0000	34.3.2.22/34-23
0x0090	Reserved			
0x0094	CTR0 — Conversion timing register 0	R/W[1]	0x0000_0203	34.3.2.23/34-24
0x0098	CTR1— Conversion timing register 1	R/W[1]	0x0000_0203	34.3.2.24/34-25
0x009C	CTR2 — Conversion timing register 2	R/W[1]	0x0000_0203	34.3.2.25/34-25
0x00A0	Reserved			
0x00A4	NCMR0 — Normal conversion mask register 0	R/W[1]	0x0000_0000	34.3.2.26/34-34
0x00A8	NCMR1 — Normal conversion mask register 1	R/W[1]	0x0000_0000	34.3.2.27/34-34
0x00AC	NCMR2 — Normal conversion mask register 2	R/W[1]	0x0000_0000	34.3.2.28/34-35
0x00B0	Reserved			
0x00B4	JCMR0 — Injected conversion mask register 0	R/W[1]	0x0000_0000	34.3.2.29/34-35

Table A-4. PXN20 Detailed Register Map (continued)

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
0x00B8	JCMR1 — Injected conversion mask register 1	R/W[1]	0x0000_0000	34.3.2.30/34-36
0x00BC	JCMR2 — Injected conversion mask register 2	R/W[1]	0x0000_0000	34.3.2.31/34-36
0x00C0	OFFWR — Offset word register	R/W[1]	0x0000_0000	34.3.2.32/34-37
0x00C4	DSD — Decode signals delay register	R/W[1]	0x0000_0000	34.3.2.33/34-38
0x00C8	PDEDR — Power down exit delay register	R/W[1]	0x0000_0000	34.3.2.34/34-38
0x00CC – 0x00FC	Reserved			
0x0100	PRECDATAREG0 — Channel 0 data register	R	0x0000_0000	34.3.2.35/34-39
0x0104	PRECDATAREG1 — Channel 1 data register	R	0x0000_0000	34.3.2.35/34-39
0x0108	PRECDATAREG2 — Channel 2 data register	R	0x0000_0000	34.3.2.35/34-39
0x010C	PRECDATAREG3 — Channel 3 data register	R	0x0000_0000	34.3.2.35/34-39
0x0110	PRECDATAREG4 — Channel 4 data register	R	0x0000_0000	34.3.2.35/34-39
0x0114	PRECDATAREG5 — Channel 5 data register	R	0x0000_0000	34.3.2.35/34-39
0x0118	PRECDATAREG6 — Channel 6 data register	R	0x0000_0000	34.3.2.35/34-39
0x011C	PRECDATAREG7 — Channel 7 data register	R	0x0000_0000	34.3.2.35/34-39
0x0120	PRECDATAREG8 — Channel 8 data register	R	0x0000_0000	34.3.2.35/34-39
0x0124	PRECDATAREG9 — Channel 9 data register	R	0x0000_0000	34.3.2.35/34-39
0x0128	PRECDATAREG10 — Channel 10 data register	R	0x0000_0000	34.3.2.35/34-39
0x012C	PRECDATAREG11 — Channel 11 data register	R	0x0000_0000	34.3.2.35/34-39
0x0130	PRECDATAREG12 — Channel 12 data register	R	0x0000_0000	34.3.2.35/34-39
0x0134	PRECDATAREG13 — Channel 13 data register	R	0x0000_0000	34.3.2.35/34-39
0x0138	PRECDATAREG14 — Channel 14 data register	R	0x0000_0000	34.3.2.35/34-39
0x013C	PRECDATAREG15 — Channel 15 data register	R	0x0000_0000	34.3.2.35/34-39
0x0140	PRECDATAREG16 — Channel 16 data register	R	0x0000_0000	34.3.2.35/34-39
0x0144	PRECDATAREG17 — Channel 17 data register	R	0x0000_0000	34.3.2.35/34-39
0x0148	PRECDATAREG18 — Channel 18 data register	R	0x0000_0000	34.3.2.35/34-39
0x014C	PRECDATAREG19 — Channel 19 data register	R	0x0000_0000	34.3.2.35/34-39
0x0150	PRECDATAREG20 — Channel 20 data register	R	0x0000_0000	34.3.2.35/34-39
0x0154	PRECDATAREG21 — Channel 21 data register	R	0x0000_0000	34.3.2.35/34-39
0x0158	PRECDATAREG22 — Channel 22 data register	R	0x0000_0000	34.3.2.35/34-39
0x015C	PRECDATAREG23 — Channel 23 data register	R	0x0000_0000	34.3.2.35/34-39
0x0160	PRECDATAREG24 — Channel 24 data register	R	0x0000_0000	34.3.2.35/34-39
0x0164	PRECDATAREG25 — Channel 25 data register	R	0x0000_0000	34.3.2.35/34-39

Table A-4. PXN20 Detailed Register Map (continued)

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
0x0168	PRECDATAREG26 — Channel 26 data register	R	0x0000_0000	34.3.2.35/34-39
0x016C	PRECDATAREG27 — Channel 27 data register	R	0x0000_0000	34.3.2.35/34-39
0x0170	PRECDATAREG28 — Channel 28 data register	R	0x0000_0000	34.3.2.35/34-39
0x0174	PRECDATAREG29 — Channel 29 data register	R	0x0000_0000	34.3.2.35/34-39
0x0178	PRECDATAREG30 — Channel 30 data register	R	0x0000_0000	34.3.2.35/34-39
0x017C	PRECDATAREG31 — Channel 31 data register	R	0x0000_0000	34.3.2.35/34-39
0x0180	INTDATAREG0 — Channel 32 data register	R	0x0000_0000	34.3.2.36/34-39
0x0184	INTDATAREG1 — Channel 33 data register	R	0x0000_0000	34.3.2.36/34-39
0x0188	INTDATAREG2 — Channel 34 data register	R	0x0000_0000	34.3.2.36/34-39
0x018C	INTDATAREG3 — Channel 35 data register	R	0x0000_0000	34.3.2.36/34-39
0x0190	INTDATAREG4 — Channel 36 data register	R	0x0000_0000	34.3.2.36/34-39
0x0194	INTDATAREG5 — Channel 37 data register	R	0x0000_0000	34.3.2.36/34-39
0x0198	INTDATAREG6 — Channel 38 data register	R	0x0000_0000	34.3.2.36/34-39
0x019C	INTDATAREG7 — Channel 39 data register	R	0x0000_0000	34.3.2.36/34-39
0x01A0	INTDATAREG8 — Channel 40 data register	R	0x0000_0000	34.3.2.36/34-39
0x01A4	INTDATAREG9 — Channel 41 data register	R	0x0000_0000	34.3.2.36/34-39
0x01A8	INTDATAREG10 — Channel 42 data register	R	0x0000_0000	34.3.2.36/34-39
0x01AC	INTDATAREG11 — Channel 43 data register	R	0x0000_0000	34.3.2.36/34-39
0x01B0	INTDATAREG12 — Channel 44 data register	R	0x0000_0000	34.3.2.36/34-39
0x01B4	INTDATAREG13 — Channel 45 data register)	R	0x0000_0000	34.3.2.36/34-39
0x01B8	INTDATAREG14 — Channel 46 data register	R	0x0000_0000	34.3.2.36/34-39
0x01BC	INTDATAREG15 — Channel 47 data register	R	0x0000_0000	34.3.2.36/34-39
0x01C0	INTDATAREG16 — Channel 48 data register	R	0x0000_0000	34.3.2.36/34-39
0x01C4	INTDATAREG17 — Channel 49 data register	R	0x0000_0000	34.3.2.36/34-39
0x01C8	INTDATAREG18 — Channel 50 data register	R	0x0000_0000	34.3.2.36/34-39
0x01CC	INTDATAREG19 — Channel 51 data register	R	0x0000_0000	34.3.2.36/34-39
0x01D0	INTDATAREG20 — Channel 52 data register	R	0x0000_0000	34.3.2.36/34-39
0x01D4	INTDATAREG21 — Channel 53 data register	R	0x0000_0000	34.3.2.36/34-39
0x01D8	INTDATAREG22 — Channel 54 data register	R	0x0000_0000	34.3.2.36/34-39
0x01DC	INTDATAREG23 — Channel 55 data register	R	0x0000_0000	34.3.2.36/34-39
0x01E0	INTDATAREG24 — Channel 56 data register	R	0x0000_0000	34.3.2.36/34-39
0x01E4	INTDATAREG25 — Channel 57 data register	R	0x0000_0000	34.3.2.36/34-39

Table A-4. PXN20 Detailed Register Map (continued)

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
0x01E8	INTDATAREG26 — Channel 58 data register	R	0x0000_0000	34.3.2.36/34-39
0x01EC	INTDATAREG27 — Channel 59 data register	R	0x0000_0000	34.3.2.36/34-39
0x01F0	INTDATAREG28 — Channel 60 data register	R	0x0000_0000	34.3.2.36/34-39
0x01F4	INTDATAREG29 — Channel 61 data register	R	0x0000_0000	34.3.2.36/34-39
0x01F8	INTDATAREG30 — Channel 62 data register	R	0x0000_0000	34.3.2.36/34-39
0x01FC	INTDATAREG31 — Channel 63 data register	R	0x0000_0000	34.3.2.36/34-39
0x0200	EXTDATAREG0 — Channel 64 data register	R	0x0000_0000	34.3.2.37/34-40
0x0204	EXTDATAREG1 — Channel 65 data register	R	0x0000_0000	34.3.2.37/34-40
0x0208	EXTDATAREG2 — Channel 66 data register	R	0x0000_0000	34.3.2.37/34-40
0x020C	EXTDATAREG3 — Channel 67 data register	R	0x0000_0000	34.3.2.37/34-40
0x0210	EXTDATAREG4 — Channel 68 data register	R	0x0000_0000	34.3.2.37/34-40
0x0214	EXTDATAREG5 — Channel 69 data register	R	0x0000_0000	34.3.2.37/34-40
0x0218	EXTDATAREG6 — Channel 70 data register	R	0x0000_0000	34.3.2.37/34-40
0x021C	EXTDATAREG7 — Channel 71 data register	R	0x0000_0000	34.3.2.37/34-40
0x0220	EXTDATAREG8 — Channel 72 data register	R	0x0000_0000	34.3.2.37/34-40
0x0224	EXTDATAREG9 — Channel 73 data register	R	0x0000_0000	34.3.2.37/34-40
0x0228	EXTDATAREG10 — Channel 74 data register	R	0x0000_0000	34.3.2.37/34-40
0x022C	EXTDATAREG11 — Channel 75 data register	R	0x0000_0000	34.3.2.37/34-40
0x0230	EXTDATAREG12 — Channel 76 data register	R	0x0000_0000	34.3.2.37/34-40
0x0234	EXTDATAREG13 — Channel 77 data register	R	0x0000_0000	34.3.2.37/34-40
0x0238	EXTDATAREG14 — Channel 78 data register	R	0x0000_0000	34.3.2.37/34-40
0x023C	EXTDATAREG15 — Channel 79 data register	R	0x0000_0000	34.3.2.37/34-40
0x0240	EXTDATAREG16 — Channel 80 data register	R	0x0000_0000	34.3.2.37/34-40
0x0244	EXTDATAREG17 — Channel 81 data register	R	0x0000_0000	34.3.2.37/34-40
0x0248	EXTDATAREG18 — Channel 82 data register	R	0x0000_0000	34.3.2.37/34-40
0x024C	EXTDATAREG19 — Channel 83 data register	R	0x0000_0000	34.3.2.37/34-40
0x0250	EXTDATAREG20 — Channel 84 data register	R	0x0000_0000	34.3.2.37/34-40
0x0254	Reserved			
0x0258	EXTDATAREG21 — Channel 85 data register	R	0x0000_0000	34.3.2.37/34-40
0x025C	EXTDATAREG22 — Channel 86 data register	R	0x0000_0000	34.3.2.37/34-40
0x0260	EXTDATAREG23 — Channel 87 data register	R	0x0000_0000	34.3.2.37/34-40
0x0264	EXTDATAREG24 — Channel 88 data register	R	0x0000_0000	34.3.2.37/34-40

Table A-4. PXN20 Detailed Register Map (continued)

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
0x0268	EXTDATAREG25 — Channel 89 data register	R	0x0000_0000	34.3.2.37/34-40
0x026C	EXTDATAREG26 — Channel 90 data register	R	0x0000_0000	34.3.2.37/34-40
0x0270	EXTDATAREG27 — Channel 91 data register	R	0x0000_0000	34.3.2.37/34-40
0x0274	EXTDATAREG28 — Channel 92 data register	R	0x0000_0000	34.3.2.37/34-40
0x0278	EXTDATAREG29 — Channel 93 data register	R	0x0000_0000	34.3.2.37/34-40
0x027C	EXTDATAREG30 — Channel 94 data register	R	0x0000_0000	34.3.2.37/34-40
0x0280	EXTDATAREG31 — Channel 95 data register	R	0x0000_0000	34.3.2.37/34-40
0x0284– 0x3FFF	Reserved			
0xFFF8_4000 – 0xFFF8_7FFF	**Reserved**			
0xFFF8_8000	**I^2C_A** **Chapter 32, Inter-Integrated Circuit Bus Controller Module (I2C)**			
0x0000	IBAD—I^2C bus address register	R/W	0x00	32.3.2.1/32-5
0x0001	IBFD—I^2C bus frequency divider register	R/W	0x00	32.3.2.2/32-5
0x0002	IBCR—I^2C bus control register	R/W	0x80	32.3.2.3/32-8
0x0003	IBSR—I^2C bus status register	R/W	0x80	32.3.2.4/32-9
0x0004	IBDR—I^2C bus data I/O register	R/W	0x00	32.3.2.5/32-10
0x0005	IBIC—I^2C bus interrupt configuration register	R/W	0x00	32.3.2.6/32-11
0x0006–0x3FFF	Reserved			
0xFFF8_C000	**I^2C_B** **Chapter 32, Inter-Integrated Circuit Bus Controller Module (I2C)**			
0x0000	IBAD—I^2C bus address register	R/W	0x00	32.3.2.1/32-5
0x0001	IBFD—I^2C bus frequency divider register	R/W	0x00	32.3.2.2/32-5
0x0002	IBCR—I^2C bus control register	R/W	0x80	32.3.2.3/32-8
0x0003	IBSR—I^2C bus status register	R/W	0x80	32.3.2.4/32-9
0x0004	IBDR—I^2C bus data I/O register	R/W	0x00	32.3.2.5/32-10
0x0005	IBIC—I^2C bus interrupt configuration register	R/W	0x00	32.3.2.6/32-11
0x0006–0x3FFF	Reserved			
0xFFF9_0000	**DSPI_A** **Chapter 30, Deserial – Serial Peripheral Interface (DSPI)**			
0x0000	DSPI_MCR—DSPI module configuration register	R/W	0x0000_4001	30.3.2.1/30-7
0x0004	Reserved			
0x0008	DSPI_TCR—DSPI transfer count register	R/W	0x0000_0000	30.3.2.2/30-9

Table A-4. PXN20 Detailed Register Map (continued)

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
0x000C	DSPI_CTAR0—DSPI clock and transfer attributes register 0	R/W	0x7800_0000	30.3.2.3/30-10
0x0010	DSPI_CTAR1—DSPI clock and transfer attributes register 1	R/W	0x7800_0000	30.3.2.3/30-10
0x0014	DSPI_CTAR2—DSPI clock and transfer attributes register 2	R/W	0x7800_0000	30.3.2.3/30-10
0x0018	DSPI_CTAR3—DSPI clock and transfer attributes register 3	R/W	0x7800_0000	30.3.2.3/30-10
0x001C	DSPI_CTAR4—DSPI clock and transfer attributes register 4	R/W	0x7800_0000	30.3.2.3/30-10
0x0020	DSPI_CTAR5—DSPI clock and transfer attributes register 5	R/W	0x7800_0000	30.3.2.3/30-10
0x0024	DSPI_CTAR6—DSPI clock and transfer attributes register 6	R/W	0x7800_0000	30.3.2.3/30-10
0x0028	DSPI_CTAR7—DSPI clock and transfer attributes register 7	R/W	0x7800_0000	30.3.2.3/30-10
0x002C	DSPI_SR—DSPI status register	R	0x0200_0000	30.3.2.4/30-16
0x0030	DSPI_RSER—DSPI DMA/interrupt request select and enable register	R/W	0x0000_0000	30.3.2.5/30-18
FIFO Registers				
0x0034	DSPI_PUSHR—DSPI push TX FIFO register	R/W	0x0000_0000	30.3.2.6/30-19
0x0038	DSPI_POPR—DSPI pop RX FIFO register	R	0x0000_0000	30.3.2.7/30-21
0x003C	DSPI_TXFR0—DSPI transmit FIFO register 0	R	0x0000_0000	30.3.2.8/30-21
0x0040	DSPI_TXFR1—DSPI transmit FIFO register 1	R	0x0000_0000	30.3.2.8/30-21
0x0044	DSPI_TXFR2—DSPI transmit FIFO register 2	R	0x0000_0000	30.3.2.8/30-21
0x0048	DSPI_TXFR3—DSPI transmit FIFO register 3	R	0x0000_0000	30.3.2.8/30-21
0x004C–0x0078	Reserved			
0x007C	DSPI_RXFR0—DSPI receive FIFO register 0	R	0x0000_0000	30.3.2.9/30-22
0x0080	DSPI_RXFR1—DSPI receive FIFO register 1	R	0x0000_0000	30.3.2.9/30-22
0x0084	DSPI_RXFR2—DSPI receive FIFO register 2	R	0x0000_0000	30.3.2.9/30-22
0x0088	DSPI_RXFR3—DSPI receive FIFO register 3	R	0x0000_0000	30.3.2.9/30-22
0x008C–0x00B8	Reserved			
0xFFF9_4000	**DSPI_B** **Chapter 30, Deserial – Serial Peripheral Interface (DSPI)**			
0x0000	DSPI_MCR—DSPI module configuration register	R/W	0x0000_4001	30.3.2.1/30-7
0x0004	Reserved			
0x0008	DSPI_TCR—DSPI transfer count register	R/W	0x0000_0000	30.3.2.2/30-9
0x000C	DSPI_CTAR0—DSPI clock and transfer attributes register 0	R/W	0x7800_0000	30.3.2.3/30-10
0x0010	DSPI_CTAR1—DSPI clock and transfer attributes register 1	R/W	0x7800_0000	30.3.2.3/30-10
0x0014	DSPI_CTAR2—DSPI clock and transfer attributes register 2	R/W	0x7800_0000	30.3.2.3/30-10

Table A-4. PXN20 Detailed Register Map (continued)

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
0x0018	DSPI_CTAR3—DSPI clock and transfer attributes register 3	R/W	0x7800_0000	30.3.2.3/30-10
0x001C	DSPI_CTAR4—DSPI clock and transfer attributes register 4	R/W	0x7800_0000	30.3.2.3/30-10
0x0020	DSPI_CTAR5—DSPI clock and transfer attributes register 5	R/W	0x7800_0000	30.3.2.3/30-10
0x0024	DSPI_CTAR6—DSPI clock and transfer attributes register 6	R/W	0x7800_0000	30.3.2.3/30-10
0x0028	DSPI_CTAR7—DSPI clock and transfer attributes register 7	R/W	0x7800_0000	30.3.2.3/30-10
0x002C	DSPI_SR—DSPI status register	R	0x0200_0000	30.3.2.4/30-16
0x0030	DSPI_RSER—DSPI DMA/interrupt request select and enable register	R/W	0x0000_0000	30.3.2.5/30-18
FIFO Registers				
0x0034	DSPI_PUSHR—DSPI push TX FIFO register	R/W	0x0000_0000	30.3.2.6/30-19
0x0038	DSPI_POPR—DSPI pop RX FIFO register	R	0x0000_0000	30.3.2.7/30-21
0x003C	DSPI_TXFR0—DSPI transmit FIFO register 0	R	0x0000_0000	30.3.2.8/30-21
0x0040	DSPI_TXFR1—DSPI transmit FIFO register 1	R	0x0000_0000	30.3.2.8/30-21
0x0044	DSPI_TXFR2—DSPI transmit FIFO register 2	R	0x0000_0000	30.3.2.8/30-21
0x0048	DSPI_TXFR3—DSPI transmit FIFO register 3	R	0x0000_0000	30.3.2.8/30-21
0x004C–0x0078	Reserved			
0x007C	DSPI_RXFR0—DSPI receive FIFO register 0	R	0x0000_0000	30.3.2.9/30-22
0x0080	DSPI_RXFR1—DSPI receive FIFO register 1	R	0x0000_0000	30.3.2.9/30-22
0x0084	DSPI_RXFR2—DSPI receive FIFO register 2	R	0x0000_0000	30.3.2.9/30-22
0x0088	DSPI_RXFR3—DSPI receive FIFO register 3	R	0x0000_0000	30.3.2.9/30-22
0x008C–0x00B8	Reserved			
0xFFF9_8000 – 0xFFF9_FFFF	**Reserved**			
0xFFFA_0000	**eSCI_A** **Chapter 31, Enhanced Serial Communication Interface (eSCI)**			
0x0000	eSCI_BRR—eSCI baud rate register	R/W	0x0004	31.3.2.1/31-6
0x0002	eSCI_CR1—eSCI control register 1	R/W	0x0000	31.3.2.2/31-6
0x0004	eSCI_CR2—eSCI control register 2	R/W	0x0200	31.3.2.3/31-8
0x0006	eSCI_SDR—eSCI data register	R/W	0x0000	31.3.2.4/31-10
0x0008	eSCI_IFSR1—eSCI interrupt flag and status register 1	R/W	0x0000	31.3.2.5/31-11
0x000A	eSCI_IFSR2—eSCI interrupt flag and status register 2	R/W	0x0000	31.3.2.6/31-12
0x000C	eSCI_LCR1—eSCI LIN control register 1	R/W	0x0000	31.3.2.7/31-13
0x000E	eSCI_LCR2—eSCI LIN control register 2	R/W	0x0000	31.3.2.8/31-15

Table A-4. PXN20 Detailed Register Map (continued)

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
0x00010	eSCI_LTR— eSCI LIN transmit register	R/W	0x0000	31.3.2.9/31-15
0x0012	Reserved			
0x0014	eSCI_LRR—eSCI LIN receive register	R/W	0x0000	31.3.2.10/31-17
0x0016	Reserved			
0x0018	eSCI_LPR—eSCI LIN CRC polynomial register	R/W	0xC599	31.3.2.11/31-18
0x001A	eSCI_CR3—eSCI control register 3	R/W	0x0000	31.3.2.12/31-18
0x001C–0x3FFF	Reserved			
0xFFFA_4000	**eSCI_B** **Chapter 31, Enhanced Serial Communication Interface (eSCI)**			
0x0000	eSCI_BRR—eSCI baud rate register	R/W	0x0004	31.3.2.1/31-6
0x0002	eSCI_CR1—eSCI control register 1	R/W	0x0000	31.3.2.2/31-6
0x0004	eSCI_CR2—eSCI control register 2	R/W	0x0200	31.3.2.3/31-8
0x0006	eSCI_SDR—eSCI data register	R/W	0x0000	31.3.2.4/31-10
0x0008	eSCI_IFSR1—eSCI interrupt flag and status register 1	R/W	0x0000	31.3.2.5/31-11
0x000A	eSCI_IFSR2—eSCI interrupt flag and status register 2	R/W	0x0000	31.3.2.6/31-12
0x000C	eSCI_LCR1—eSCI LIN control register 1	R/W	0x0000	31.3.2.7/31-13
0x000E	eSCI_LCR2—eSCI LIN control register 2	R/W	0x0000	31.3.2.8/31-15
0x00010	eSCI_LTR— eSCI LIN transmit register	R/W	0x0000	31.3.2.9/31-15
0x0012	Reserved			
0x0014	eSCI_LRR—eSCI LIN receive register	R/W	0x0000	31.3.2.10/31-17
0x0016	Reserved			
0x0018	eSCI_LPR—eSCI LIN CRC polynomial register	R/W	0xC599	31.3.2.11/31-18
0x001A	eSCI_CR3—eSCI control register 3	R/W	0x0000	31.3.2.12/31-18
0x001C–0x3FFF	Reserved			
0xFFFA_8000	**eSCI_C** **Chapter 31, Enhanced Serial Communication Interface (eSCI)**			
0x0000	eSCI_BRR—eSCI baud rate register	R/W	0x0004	31.3.2.1/31-6
0x0002	eSCI_CR1—eSCI control register 1	R/W	0x0000	31.3.2.2/31-6
0x0004	eSCI_CR2—eSCI control register 2	R/W	0x0200	31.3.2.3/31-8
0x0006	eSCI_SDR—eSCI data register	R/W	0x0000	31.3.2.4/31-10
0x0008	eSCI_IFSR1—eSCI interrupt flag and status register 1	R/W	0x0000	31.3.2.5/31-11
0x000A	eSCI_IFSR2—eSCI interrupt flag and status register 2	R/W	0x0000	31.3.2.6/31-12

Table A-4. PXN20 Detailed Register Map (continued)

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
0x000C	eSCI_LCR1—eSCI LIN control register 1	R/W	0x0000	31.3.2.7/31-13
0x000E	eSCI_LCR2—eSCI LIN control register 2	R/W	0x0000	31.3.2.8/31-15
0x00010	eSCI_LTR— eSCI LIN transmit register	R/W	0x0000	31.3.2.9/31-15
0x0012	Reserved			
0x0014	eSCI_LRR—eSCI LIN receive register	R/W	0x0000	31.3.2.10/31-17
0x0016	Reserved			
0x0018	eSCI_LPR—eSCI LIN CRC polynomial register	R/W	0xC599	31.3.2.11/31-18
0x001A	eSCI_CR3—eSCI control register 3	R/W	0x0000	31.3.2.12/31-18
0x001C–0x3FFF	Reserved			
0xFFFA_C000	**eSCI_D** **Chapter 31, Enhanced Serial Communication Interface (eSCI)**			
0x0000	eSCI_BRR—eSCI baud rate register	R/W	0x0004	31.3.2.1/31-6
0x0002	eSCI_CR1—eSCI control register 1	R/W	0x0000	31.3.2.2/31-6
0x0004	eSCI_CR2—eSCI control register 2	R/W	0x0200	31.3.2.3/31-8
0x0006	eSCI_SDR—eSCI data register	R/W	0x0000	31.3.2.4/31-10
0x0008	eSCI_IFSR1—eSCI interrupt flag and status register 1	R/W	0x0000	31.3.2.5/31-11
0x000A	eSCI_IFSR2—eSCI interrupt flag and status register 2	R/W	0x0000	31.3.2.6/31-12
0x000C	eSCI_LCR1—eSCI LIN control register 1	R/W	0x0000	31.3.2.7/31-13
0x000E	eSCI_LCR2—eSCI LIN control register 2	R/W	0x0000	31.3.2.8/31-15
0x00010	eSCI_LTR— eSCI LIN transmit register	R/W	0x0000	31.3.2.9/31-15
0x0012	Reserved			
0x0014	eSCI_LRR—eSCI LIN receive register	R/W	0x0000	31.3.2.10/31-17
0x0016	Reserved			
0x0018	eSCI_LPR—eSCI LIN CRC polynomial register	R/W	0xC599	31.3.2.11/31-18
0x001A	eSCI_CR3—eSCI control register 3	R/W	0x0000	31.3.2.12/31-18
0x001C–0x3FFF	Reserved			
0xFFFB_0000	**eSCI_E** **Chapter 31, Enhanced Serial Communication Interface (eSCI)**			
0x0000	eSCI_BRR—eSCI baud rate register	R/W	0x0004	31.3.2.1/31-6
0x0002	eSCI_CR1—eSCI control register 1	R/W	0x0000	31.3.2.2/31-6
0x0004	eSCI_CR2—eSCI control register 2	R/W	0x0200	31.3.2.3/31-8
0x0006	eSCI_SDR—eSCI data register	R/W	0x0000	31.3.2.4/31-10

Table A-4. PXN20 Detailed Register Map (continued)

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
0x0008	eSCI_IFSR1—eSCI interrupt flag and status register 1	R/W	0x0000	31.3.2.5/31-11
0x000A	eSCI_IFSR2—eSCI interrupt flag and status register 2	R/W	0x0000	31.3.2.6/31-12
0x000C	eSCI_LCR1—eSCI LIN control register 1	R/W	0x0000	31.3.2.7/31-13
0x000E	eSCI_LCR2—eSCI LIN control register 2	R/W	0x0000	31.3.2.8/31-15
0x00010	eSCI_LTR— eSCI LIN transmit register	R/W	0x0000	31.3.2.9/31-15
0x0012	Reserved			
0x0014	eSCI_LRR—eSCI LIN receive register	R/W	0x0000	31.3.2.10/31-17
0x0016	Reserved			
0x0018	eSCI_LPR—eSCI LIN CRC polynomial register	R/W	0xC599	31.3.2.11/31-18
0x001A	eSCI_CR3—eSCI control register 3	R/W	0x0000	31.3.2.12/31-18
0x001C–0x3FFF	Reserved			
0xFFFB_4000	**eSCI_F** **Chapter 31, Enhanced Serial Communication Interface (eSCI)**			
0x0000	eSCI_BRR—eSCI baud rate register	R/W	0x0004	31.3.2.1/31-6
0x0002	eSCI_CR1—eSCI control register 1	R/W	0x0000	31.3.2.2/31-6
0x0004	eSCI_CR2—eSCI control register 2	R/W	0x0200	31.3.2.3/31-8
0x0006	eSCI_SDR—eSCI data register	R/W	0x0000	31.3.2.4/31-10
0x0008	eSCI_IFSR1—eSCI interrupt flag and status register 1	R/W	0x0000	31.3.2.5/31-11
0x000A	eSCI_IFSR2—eSCI interrupt flag and status register 2	R/W	0x0000	31.3.2.6/31-12
0x000C	eSCI_LCR1—eSCI LIN control register 1	R/W	0x0000	31.3.2.7/31-13
0x000E	eSCI_LCR2—eSCI LIN control register 2	R/W	0x0000	31.3.2.8/31-15
0x00010	eSCI_LTR— eSCI LIN transmit register	R/W	0x0000	31.3.2.9/31-15
0x0012	Reserved			
0x0014	eSCI_LRR—eSCI LIN receive register	R/W	0x0000	31.3.2.10/31-17
0x0016	Reserved			
0x0018	eSCI_LPR—eSCI LIN CRC polynomial register	R/W	0xC599	31.3.2.11/31-18
0x001A	eSCI_CR3—eSCI control register 3	R/W	0x0000	31.3.2.12/31-18
0x001C–0x3FFF	Reserved			
0xFFFB_8000	**eSCI_G** **Chapter 31, Enhanced Serial Communication Interface (eSCI)**			
0x0000	eSCI_BRR—eSCI baud rate register	R/W	0x0004	31.3.2.1/31-6
0x0002	eSCI_CR1—eSCI control register 1	R/W	0x0000	31.3.2.2/31-6

Table A-4. PXN20 Detailed Register Map (continued)

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
0x0004	eSCI_CR2—eSCI control register 2	R/W	0x0200	31.3.2.3/31-8
0x0006	eSCI_SDR—eSCI data register	R/W	0x0000	31.3.2.4/31-10
0x0008	eSCI_IFSR1—eSCI interrupt flag and status register 1	R/W	0x0000	31.3.2.5/31-11
0x000A	eSCI_IFSR2—eSCI interrupt flag and status register 2	R/W	0x0000	31.3.2.6/31-12
0x000C	eSCI_LCR1—eSCI LIN control register 1	R/W	0x0000	31.3.2.7/31-13
0x000E	eSCI_LCR2—eSCI LIN control register 2	R/W	0x0000	31.3.2.8/31-15
0x00010	eSCI_LTR— eSCI LIN transmit register	R/W	0x0000	31.3.2.9/31-15
0x0012	Reserved			
0x0014	eSCI_LRR—eSCI LIN receive register	R/W	0x0000	31.3.2.10/31-17
0x0016	Reserved			
0x0018	eSCI_LPR—eSCI LIN CRC polynomial register	R/W	0xC599	31.3.2.11/31-18
0x001A	eSCI_CR3—eSCI control register 3	R/W	0x0000	31.3.2.12/31-18
0x001C–0x3FFF	Reserved			
0xFFFB_C000	**eSCI_H** **Chapter 31, Enhanced Serial Communication Interface (eSCI)**			
0x0000	eSCI_BRR—eSCI baud rate register	R/W	0x0004	31.3.2.1/31-6
0x0002	eSCI_CR1—eSCI control register 1	R/W	0x0000	31.3.2.2/31-6
0x0004	eSCI_CR2—eSCI control register 2	R/W	0x0200	31.3.2.3/31-8
0x0006	eSCI_SDR—eSCI data register	R/W	0x0000	31.3.2.4/31-10
0x0008	eSCI_IFSR1—eSCI interrupt flag and status register 1	R/W	0x0000	31.3.2.5/31-11
0x000A	eSCI_IFSR2—eSCI interrupt flag and status register 2	R/W	0x0000	31.3.2.6/31-12
0x000C	eSCI_LCR1—eSCI LIN control register 1	R/W	0x0000	31.3.2.7/31-13
0x000E	eSCI_LCR2—eSCI LIN control register 2	R/W	0x0000	31.3.2.8/31-15
0x00010	eSCI_LTR— eSCI LIN transmit register	R/W	0x0000	31.3.2.9/31-15
0x0012	Reserved			
0x0014	eSCI_LRR—eSCI LIN receive register	R/W	0x0000	31.3.2.10/31-17
0x0016	Reserved			
0x0018	eSCI_LPR—eSCI LIN CRC polynomial register	R/W	0xC599	31.3.2.11/31-18
0x001A	eSCI_CR3—eSCI control register 3	R/W	0x0000	31.3.2.12/31-18
0x001C–0x3FFF	Reserved			
0xFFFC_0000	**FlexCAN_A** **Chapter 29, Controller Area Network (FlexCAN)**			

Table A-4. PXN20 Detailed Register Map (continued)

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
0x0000	CANA_MCR—Module configuration register	R/W	0x9890_000F	29.3.4.1/29-11
0x0004	CANA_CTRL—control register	R/W	0x0000_0000	29.3.4.2/29-14
0x0008	CANA_TIMER—Free-running timer	R/W	0x0000_0000	29.3.4.3/29-17
0x000C	Reserved			
0x0010	CANA_RXGMASK—Rx global mask	R/W	0xFFFF_FFFF	29.3.4.4.1/29-18
0x0014	CANA_RX14MASK—Rx buffer 14 mask	R/W	0xFFFF_FFFF	29.3.4.4.2/29-19
0x0018	CANA_RX15MASK—Rx buffer 15 mask	R/W	0xFFFF_FFFF	29.3.4.4.3/29-19
0x001C	CANA_ECR—Error counter register	R/W	0x0000_0000	29.3.2/29-7
0x0020	CANA_ESR—Error and status register	R/W	0x0000_0000	29.3.4.6/29-21
0x0024	CANA_IMASK2—Interrupt masks 2	R/W	0x0000_0000	29.3.4.7/29-23
0x0028	CANA_IMASK1—Interrupt masks 1	R/W	0x0000_0000	29.3.4.8/29-24
0x002C	CANA_IFLAG2—Interrupt flags 2	R/W	0x0000_0000	29.3.4.9/29-24
0x0030	CANA_IFLAG1—Interrupt flags 1	R/W	0x0000_0000	29.3.4.10/29-25
0x0034–0x007F	Reserved			
0x0080	MB0—Message buffer 0	R/W	0x0000_0000	29.3.2/29-7
0x0090	MB1—Message buffer 1	R/W	0x0000_0000	29.3.2/29-7
0x00AO	MB2—Message buffer 2	R/W	0x0000_0000	29.3.2/29-7
0x00B0	MB3—Message buffer 3	R/W	0x0000_0000	29.3.2/29-7
0x00C0	MB4—Message buffer 4	R/W	0x0000_0000	29.3.2/29-7
0x00D0	MB5—Message buffer 5	R/W	0x0000_0000	29.3.2/29-7
0x00E0	MB6—Message buffer 6	R/W	0x0000_0000	29.3.2/29-7
0x00F0	MB7—Message buffer 7	R/W	0x0000_0000	29.3.2/29-7
0x0100	MB8—Message buffer 8	R/W	0x0000_0000	29.3.2/29-7
0x0110	MB9—Message buffer 9	R/W	0x0000_0000	29.3.2/29-7
0x0120	MB10—Message buffer 10	R/W	0x0000_0000	29.3.2/29-7
0x0130	MB11—Message buffer 11	R/W	0x0000_0000	29.3.2/29-7
0x0140	MB12—Message buffer 12	R/W	0x0000_0000	29.3.2/29-7
0x0150	MB13—Message buffer 13	R/W	0x0000_0000	29.3.2/29-7
0x0160	MB14—Message buffer 14	R/W	0x0000_0000	29.3.2/29-7
0x0170	MB15—Message buffer 15	R/W	0x0000_0000	29.3.2/29-7

Table A-4. PXN20 Detailed Register Map (continued)

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
0x0180	MB16—Message buffer 16	R/W	0x0000_0000	29.3.2/29-7
0x0190	MB17—Message buffer 17	R/W	0x0000_0000	29.3.2/29-7
0x01A0	MB18—Message buffer 18	R/W	0x0000_0000	29.3.2/29-7
0x01B0	MB19—Message buffer 19	R/W	0x0000_0000	29.3.2/29-7
0x01C0	MB20—Message buffer 20	R/W	0x0000_0000	29.3.2/29-7
0x01D0	MB21—Message buffer 21	R/W	0x0000_0000	29.3.2/29-7
0x01E0	MB22—Message buffer 22	R/W	0x0000_0000	29.3.2/29-7
0x01F0	MB23—Message buffer 23	R/W	0x0000_0000	29.3.2/29-7
0x0200	MB24—Message buffer 24	R/W	0x0000_0000	29.3.2/29-7
0x0210	MB25—Message buffer 25	R/W	0x0000_0000	29.3.2/29-7
0x0220	MB26—Message buffer 26	R/W	0x0000_0000	29.3.2/29-7
0x0230	MB27—Message buffer 27	R/W	0x0000_0000	29.3.2/29-7
0x0240	MB28—Message buffer 28	R/W	0x0000_0000	29.3.2/29-7
0x0250	MB29—Message buffer 29	R/W	0x0000_0000	29.3.2/29-7
0x0260	MB30—Message buffer 30	R/W	0x0000_0000	29.3.2/29-7
0x0270	MB31—Message buffer 31	R/W	0x0000_0000	29.3.2/29-7
0x0280	MB32—Message buffer 32	R/W	0x0000_0000	29.3.2/29-7
0x0290	MB33—Message buffer 33	R/W	0x0000_0000	29.3.2/29-7
0x02A0	MB34—Message buffer 34	R/W	0x0000_0000	29.3.2/29-7
0x02B0	MB35—Message buffer 35	R/W	0x0000_0000	29.3.2/29-7
0x02C0	MB36—Message buffer 36	R/W	0x0000_0000	29.3.2/29-7
0x02D0	MB37—Message buffer 37	R/W	0x0000_0000	29.3.2/29-7
0x02E0	MB38—Message buffer 38	R/W	0x0000_0000	29.3.2/29-7
0x02F0	MB39—Message buffer 39	R/W	0x0000_0000	29.3.2/29-7
0x0300	MB40—Message buffer 40	R/W	0x0000_0000	29.3.2/29-7
0x0310	MB41—Message buffer 41	R/W	0x0000_0000	29.3.2/29-7
0x0320	MB42—Message buffer 42	R/W	0x0000_0000	29.3.2/29-7
0x0330	MB43—Message buffer 43	R/W	0x0000_0000	29.3.2/29-7
0x0340	MB44—Message buffer 44	R/W	0x0000_0000	29.3.2/29-7
0x0350	MB45—Message buffer 45	R/W	0x0000_0000	29.3.2/29-7
0x0360	MB46—Message buffer 46	R/W	0x0000_0000	29.3.2/29-7
0x0370	MB47—Message buffer 47	R/W	0x0000_0000	29.3.2/29-7

Table A-4. PXN20 Detailed Register Map (continued)

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
0x0380	MB48—Message buffer 48	R/W	0x0000_0000	29.3.2/29-7
0x0390	MB49—Message buffer 49	R/W	0x0000_0000	29.3.2/29-7
0x03A0	MB50—Message buffer 50	R/W	0x0000_0000	29.3.2/29-7
0x03B0	MB51—Message buffer 51	R/W	0x0000_0000	29.3.2/29-7
0x03C0	MB52—Message buffer 52	R/W	0x0000_0000	29.3.2/29-7
0x03D0	MB53—Message buffer 53	R/W	0x0000_0000	29.3.2/29-7
0x03E0	MB54—Message buffer 54	R/W	0x0000_0000	29.3.2/29-7
0x03F0	MB55—Message buffer 55	R/W	0x0000_0000	29.3.2/29-7
0x0400	MB56—Message buffer 56	R/W	0x0000_0000	29.3.2/29-7
0x0410	MB57—Message buffer 57	R/W	0x0000_0000	29.3.2/29-7
0x0420	MB58—Message buffer 58	R/W	0x0000_0000	29.3.2/29-7
0x0430	MB59—Message buffer 59	R/W	0x0000_0000	29.3.2/29-7
0x0440	MB60—Message buffer 60	R/W	0x0000_0000	29.3.2/29-7
0x0450	MB61—Message buffer 61	R/W	0x0000_0000	29.3.2/29-7
0x0460	MB62—Message buffer 62	R/W	0x0000_0000	29.3.2/29-7
0x0470	MB63—Message buffer 63	R/W	0x0000_0000	29.3.2/29-7
0x0480–087F	Reserved			
0x0880	CANA_RXIMR0—Rx individual mask register 0	R/W	0x0000_0000	29.3.4.11/29-26
0x0884	CANA_RXIMR1—Rx individual mask register 1	R/W	0x0000_0000	29.3.4.11/29-26
0x0888	CANA_RXIMR2—Rx individual mask register 2	R/W	0x0000_0000	29.3.4.11/29-26
0x088C	CANA_RXIMR3—Rx individual mask register 3	R/W	0x0000_0000	29.3.4.11/29-26
0x0890	CANA_RXIMR4—Rx individual mask register 4	R/W	0x0000_0000	29.3.4.11/29-26
0x0894	CANA_RXIMR5—Rx individual mask register 5	R/W	0x0000_0000	29.3.4.11/29-26
0x0898	CANA_RXIMR6—Rx individual mask register 6	R/W	0x0000_0000	29.3.4.11/29-26
0x089C	CANA_RXIMR7—Rx individual mask register 7	R/W	0x0000_0000	29.3.4.11/29-26
0x08A0	CANA_RXIMR8—Rx individual mask register 8	R/W	0x0000_0000	29.3.4.11/29-26
0x08A4	CANA_RXIMR9—Rx individual mask register 9	R/W	0x0000_0000	29.3.4.11/29-26
0x08A8	CANA_RXIMR10—Rx individual mask register 10	R/W	0x0000_0000	29.3.4.11/29-26
0x08AC	CANA_RXIMR11—Rx individual mask register 11	R/W	0x0000_0000	29.3.4.11/29-26
0x08B0	CANA_RXIMR12—Rx individual mask register 12	R/W	0x0000_0000	29.3.4.11/29-26
0x08B4	CANA_RXIMR13—Rx individual mask register 13	R/W	0x0000_0000	29.3.4.11/29-26
0x08B8	CANA_RXIMR14—Rx individual mask register 14	R/W	0x0000_0000	29.3.4.11/29-26

Table A-4. PXN20 Detailed Register Map (continued)

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
0x08BC	CANA_RXIMR15—Rx individual mask register 15	R/W	0x0000_0000	29.3.4.11/29-26
0x08C0	CANA_RXIMR16—Rx individual mask register 16	R/W	0x0000_0000	29.3.4.11/29-26
0x08C4	CANA_RXIMR17—Rx individual mask register 17	R/W	0x0000_0000	29.3.4.11/29-26
0x08C8	CANA_RXIMR18—Rx individual mask register 18	R/W	0x0000_0000	29.3.4.11/29-26
0x08CC	CANA_RXIMR19—Rx individual mask register 19	R/W	0x0000_0000	29.3.4.11/29-26
0x08D0	CANA_RXIMR20—Rx individual mask register 20	R/W	0x0000_0000	29.3.4.11/29-26
0x08D4	CANA_RXIMR21—Rx individual mask register 21	R/W	0x0000_0000	29.3.4.11/29-26
0x08D8	CANA_RXIMR22—Rx individual mask register 22	R/W	0x0000_0000	29.3.4.11/29-26
0x08DC	CANA_RXIMR23—Rx individual mask register 23	R/W	0x0000_0000	29.3.4.11/29-26
0x08E0	CANA_RXIMR24—Rx individual mask register 24	R/W	0x0000_0000	29.3.4.11/29-26
0x08E4	CANA_RXIMR25—Rx individual mask register 25	R/W	0x0000_0000	29.3.4.11/29-26
0x08E8	CANA_RXIMR26—Rx individual mask register 26	R/W	0x0000_0000	29.3.4.11/29-26
0x08EC	CANA_RXIMR27—Rx individual mask register 27	R/W	0x0000_0000	29.3.4.11/29-26
0x08F0	CANA_RXIMR28—Rx individual mask register 28	R/W	0x0000_0000	29.3.4.11/29-26
0x08F4	CANA_RXIMR29—Rx individual mask register 29	R/W	0x0000_0000	29.3.4.11/29-26
0x08F8	CANA_RXIMR30—Rx individual mask register 30	R/W	0x0000_0000	29.3.4.11/29-26
0x08FC	CANA_RXIMR31—Rx individual mask register 31	R/W	0x0000_0000	29.3.4.11/29-26
0x0900	CANA_RXIMR32—Rx individual mask register 32	R/W	0x0000_0000	29.3.4.11/29-26
0x0904	CANA_RXIMR33—Rx individual mask register 33	R/W	0x0000_0000	29.3.4.11/29-26
0x0908	CANA_RXIMR34—Rx individual mask register 34	R/W	0x0000_0000	29.3.4.11/29-26
0x090C	CANA_RXIMR35—Rx individual mask register 35	R/W	0x0000_0000	29.3.4.11/29-26
0x0910	CANA_RXIMR36—Rx individual mask register 36	R/W	0x0000_0000	29.3.4.11/29-26
0x0914	CANA_RXIMR37—Rx individual mask register 37	R/W	0x0000_0000	29.3.4.11/29-26
0x0918	CANA_RXIMR38—Rx individual mask register 38	R/W	0x0000_0000	29.3.4.11/29-26
0x091C	CANA_RXIMR39—Rx individual mask register 39	R/W	0x0000_0000	29.3.4.11/29-26
0x0920	CANA_RXIMR40—Rx individual mask register 40	R/W	0x0000_0000	29.3.4.11/29-26
0x0924	CANA_RXIMR41—Rx individual mask register 41	R/W	0x0000_0000	29.3.4.11/29-26
0x0928	CANA_RXIMR42—Rx individual mask register 42	R/W	0x0000_0000	29.3.4.11/29-26
0x092C	CANA_RXIMR43—Rx individual mask register 43	R/W	0x0000_0000	29.3.4.11/29-26
0x0930	CANA_RXIMR44—Rx individual mask register 44	R/W	0x0000_0000	29.3.4.11/29-26
0x0934	CANA_RXIMR45—Rx individual mask register 45	R/W	0x0000_0000	29.3.4.11/29-26
0x0938	CANA_RXIMR46—Rx individual mask register 46	R/W	0x0000_0000	29.3.4.11/29-26

Table A-4. PXN20 Detailed Register Map (continued)

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
0x093C	CANA_RXIMR47—Rx individual mask register 47	R/W	0x0000_0000	29.3.4.11/29-26
0x0940	CANA_RXIMR48—Rx individual mask register 48	R/W	0x0000_0000	29.3.4.11/29-26
0x0944	CANA_RXIMR49—Rx individual mask register 49	R/W	0x0000_0000	29.3.4.11/29-26
0x0948	CANA_RXIMR50—Rx individual mask register 50	R/W	0x0000_0000	29.3.4.11/29-26
0x094C	CANA_RXIMR51—Rx individual mask register 51	R/W	0x0000_0000	29.3.4.11/29-26
0x0950	CANA_RXIMR52—Rx individual mask register 52	R/W	0x0000_0000	29.3.4.11/29-26
0x0954	CANA_RXIMR53—Rx individual mask register 53	R/W	0x0000_0000	29.3.4.11/29-26
0x0958	CANA_RXIMR54—Rx individual mask register 54	R/W	0x0000_0000	29.3.4.11/29-26
0x095C	CANA_RXIMR55—Rx individual mask register 55	R/W	0x0000_0000	29.3.4.11/29-26
0x0960	CANA_RXIMR56—Rx individual mask register 56	R/W	0x0000_0000	29.3.4.11/29-26
0x0964	CANA_RXIMR57—Rx individual mask register 57	R/W	0x0000_0000	29.3.4.11/29-26
0x0968	CANA_RXIMR58—Rx individual mask register 58	R/W	0x0000_0000	29.3.4.11/29-26
0x096C	CANA_RXIMR59—Rx individual mask register 59	R/W	0x0000_0000	29.3.4.11/29-26
0x0970	CANA_RXIMR60—Rx individual mask register 60	R/W	0x0000_0000	29.3.4.11/29-26
0x0974	CANA_RXIMR61—Rx individual mask register 61	R/W	0x0000_0000	29.3.4.11/29-26
0x0978	CANA_RXIMR62—Rx individual mask register 62	R/W	0x0000_0000	29.3.4.11/29-26
0x097C	CANA_RXIMR63—Rx individual mask register 63	R/W	0x0000_0000	29.3.4.11/29-26
0x0980–0x3FFF	Reserved			
0xFFFC_4000	**FlexCAN_B** **Chapter 29, Controller Area Network (FlexCAN)**			
0x0000	CANB_MCR—Module configuration register	R/W	0x9890_000F	29.3.4.1/29-11
0x0004	CANB_CTRL—control register	R/W	0x0000_0000	29.3.4.2/29-14
0x0008	CANB_TIMER—Free-running timer	R/W	0x0000_0000	29.3.4.3/29-17
0x000C	Reserved			
0x0010	CANB_RXGMASK—Rx global mask	R/W	0xFFFF_FFFF	29.3.4.4.1/29-18
0x0014	CANB_RX14MASK—Rx buffer 14 mask	R/W	0xFFFF_FFFF	29.3.4.4.2/29-19
0x0018	CANB_RX15MASK—Rx buffer 15 mask	R/W	0xFFFF_FFFF	29.3.4.4.3/29-19
0x001C	CANB_ECR—Error counter register	R/W	0x0000_0000	29.3.2/29-7
0x0020	CANB_ESR—Error and status register	R/W	0x0000_0000	29.3.4.6/29-21
0x0024	CANB_IMASK2—Interrupt masks 2	R/W	0x0000_0000	29.3.4.7/29-23

Table A-4. PXN20 Detailed Register Map (continued)

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
0x0028	CANB_IMASK1—Interrupt masks 1	R/W	0x0000_0000	29.3.4.8/29-24
0x002C	CANB_IFLAG2—Interrupt flags 2	R/W	0x0000_0000	29.3.4.9/29-24
0x0030	CANB_IFLAG1—Interrupt flags 1	R/W	0x0000_0000	29.3.4.10/29-25
0x0034–0x007F	Reserved			
0x0080	MB0—Message buffer 0	R/W	0x0000_0000	29.3.2/29-7
0x0090	MB1—Message buffer 1	R/W	0x0000_0000	29.3.2/29-7
0x00AO	MB2—Message buffer 2	R/W	0x0000_0000	29.3.2/29-7
0x00B0	MB3—Message buffer 3	R/W	0x0000_0000	29.3.2/29-7
0x00C0	MB4—Message buffer 4	R/W	0x0000_0000	29.3.2/29-7
0x00D0	MB5—Message buffer 5	R/W	0x0000_0000	29.3.2/29-7
0x00E0	MB6—Message buffer 6	R/W	0x0000_0000	29.3.2/29-7
0x00F0	MB7—Message buffer 7	R/W	0x0000_0000	29.3.2/29-7
0x0100	MB8—Message buffer 8	R/W	0x0000_0000	29.3.2/29-7
0x0110	MB9—Message buffer 9	R/W	0x0000_0000	29.3.2/29-7
0x0120	MB10—Message buffer 10	R/W	0x0000_0000	29.3.2/29-7
0x0130	MB11—Message buffer 11	R/W	0x0000_0000	29.3.2/29-7
0x0140	MB12—Message buffer 12	R/W	0x0000_0000	29.3.2/29-7
0x0150	MB13—Message buffer 13	R/W	0x0000_0000	29.3.2/29-7
0x0160	MB14—Message buffer 14	R/W	0x0000_0000	29.3.2/29-7
0x0170	MB15—Message buffer 15	R/W	0x0000_0000	29.3.2/29-7
0x0180	MB16—Message buffer 16	R/W	0x0000_0000	29.3.2/29-7
0x0190	MB17—Message buffer 17	R/W	0x0000_0000	29.3.2/29-7
0x01A0	MB18—Message buffer 18	R/W	0x0000_0000	29.3.2/29-7
0x01B0	MB19—Message buffer 19	R/W	0x0000_0000	29.3.2/29-7
0x01C0	MB20—Message buffer 20	R/W	0x0000_0000	29.3.2/29-7
0x01D0	MB21—Message buffer 21	R/W	0x0000_0000	29.3.2/29-7
0x01E0	MB22—Message buffer 22	R/W	0x0000_0000	29.3.2/29-7
0x01F0	MB23—Message buffer 23	R/W	0x0000_0000	29.3.2/29-7
0x0200	MB24—Message buffer 24	R/W	0x0000_0000	29.3.2/29-7
0x0210	MB25—Message buffer 25	R/W	0x0000_0000	29.3.2/29-7
0x0220	MB26—Message buffer 26	R/W	0x0000_0000	29.3.2/29-7
0x0230	MB27—Message buffer 27	R/W	0x0000_0000	29.3.2/29-7

Table A-4. PXN20 Detailed Register Map (continued)

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
0x0240	MB28—Message buffer 28	R/W	0x0000_0000	29.3.2/29-7
0x0250	MB29—Message buffer 29	R/W	0x0000_0000	29.3.2/29-7
0x0260	MB30—Message buffer 30	R/W	0x0000_0000	29.3.2/29-7
0x0270	MB31—Message buffer 31	R/W	0x0000_0000	29.3.2/29-7
0x0280	MB32—Message buffer 32	R/W	0x0000_0000	29.3.2/29-7
0x0290	MB33—Message buffer 33	R/W	0x0000_0000	29.3.2/29-7
0x02A0	MB34—Message buffer 34	R/W	0x0000_0000	29.3.2/29-7
0x02B0	MB35—Message buffer 35	R/W	0x0000_0000	29.3.2/29-7
0x02C0	MB36—Message buffer 36	R/W	0x0000_0000	29.3.2/29-7
0x02D0	MB37—Message buffer 37	R/W	0x0000_0000	29.3.2/29-7
0x02E0	MB38—Message buffer 38	R/W	0x0000_0000	29.3.2/29-7
0x02F0	MB39—Message buffer 39	R/W	0x0000_0000	29.3.2/29-7
0x0300	MB40—Message buffer 40	R/W	0x0000_0000	29.3.2/29-7
0x0310	MB41—Message buffer 41	R/W	0x0000_0000	29.3.2/29-7
0x0320	MB42—Message buffer 42	R/W	0x0000_0000	29.3.2/29-7
0x0330	MB43—Message buffer 43	R/W	0x0000_0000	29.3.2/29-7
0x0340	MB44—Message buffer 44	R/W	0x0000_0000	29.3.2/29-7
0x0350	MB45—Message buffer 45	R/W	0x0000_0000	29.3.2/29-7
0x0360	MB46—Message buffer 46	R/W	0x0000_0000	29.3.2/29-7
0x0370	MB47—Message buffer 47	R/W	0x0000_0000	29.3.2/29-7
0x0380	MB48—Message buffer 48	R/W	0x0000_0000	29.3.2/29-7
0x0390	MB49—Message buffer 49	R/W	0x0000_0000	29.3.2/29-7
0x03A0	MB50—Message buffer 50	R/W	0x0000_0000	29.3.2/29-7
0x03B0	MB51—Message buffer 51	R/W	0x0000_0000	29.3.2/29-7
0x03C0	MB52—Message buffer 52	R/W	0x0000_0000	29.3.2/29-7
0x03D0	MB53—Message buffer 53	R/W	0x0000_0000	29.3.2/29-7
0x03E0	MB54—Message buffer 54	R/W	0x0000_0000	29.3.2/29-7
0x03F0	MB55—Message buffer 55	R/W	0x0000_0000	29.3.2/29-7
0x0400	MB56—Message buffer 56	R/W	0x0000_0000	29.3.2/29-7
0x0410	MB57—Message buffer 57	R/W	0x0000_0000	29.3.2/29-7
0x0420	MB58—Message buffer 58	R/W	0x0000_0000	29.3.2/29-7
0x0430	MB59—Message buffer 59	R/W	0x0000_0000	29.3.2/29-7

Table A-4. PXN20 Detailed Register Map (continued)

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
0x0440	MB60—Message buffer 60	R/W	0x0000_0000	29.3.2/29-7
0x0450	MB61—Message buffer 61	R/W	0x0000_0000	29.3.2/29-7
0x0460	MB62—Message buffer 62	R/W	0x0000_0000	29.3.2/29-7
0x0470	MB63—Message buffer 63	R/W	0x0000_0000	29.3.2/29-7
0x0480–087F	Reserved			
0x0880	CANB_RXIMR0—Rx individual mask register 0	R/W	0x0000_0000	29.3.4.11/29-26
0x0884	CANB_RXIMR1—Rx individual mask register 1	R/W	0x0000_0000	29.3.4.11/29-26
0x0888	CANB_RXIMR2—Rx individual mask register 2	R/W	0x0000_0000	29.3.4.11/29-26
0x088C	CANB_RXIMR3—Rx individual mask register 3	R/W	0x0000_0000	29.3.4.11/29-26
0x0890	CANB_RXIMR4—Rx individual mask register 4	R/W	0x0000_0000	29.3.4.11/29-26
0x0894	CANB_RXIMR5—Rx individual mask register 5	R/W	0x0000_0000	29.3.4.11/29-26
0x0898	CANB_RXIMR6—Rx individual mask register 6	R/W	0x0000_0000	29.3.4.11/29-26
0x089C	CANB_RXIMR7—Rx individual mask register 7	R/W	0x0000_0000	29.3.4.11/29-26
0x08A0	CANB_RXIMR8—Rx individual mask register 8	R/W	0x0000_0000	29.3.4.11/29-26
0x08A4	CANB_RXIMR9—Rx individual mask register 9	R/W	0x0000_0000	29.3.4.11/29-26
0x08A8	CANB_RXIMR10—Rx individual mask register 10	R/W	0x0000_0000	29.3.4.11/29-26
0x08AC	CANB_RXIMR11—Rx individual mask register 11	R/W	0x0000_0000	29.3.4.11/29-26
0x08B0	CANB_RXIMR12—Rx individual mask register 12	R/W	0x0000_0000	29.3.4.11/29-26
0x08B4	CANB_RXIMR13—Rx individual mask register 13	R/W	0x0000_0000	29.3.4.11/29-26
0x08B8	CANB_RXIMR14—Rx individual mask register 14	R/W	0x0000_0000	29.3.4.11/29-26
0x08BC	CANB_RXIMR15—Rx individual mask register 15	R/W	0x0000_0000	29.3.4.11/29-26
0x08C0	CANB_RXIMR16—Rx individual mask register 16	R/W	0x0000_0000	29.3.4.11/29-26
0x08C4	CANB_RXIMR17—Rx individual mask register 17	R/W	0x0000_0000	29.3.4.11/29-26
0x08C8	CANB_RXIMR18—Rx individual mask register 18	R/W	0x0000_0000	29.3.4.11/29-26
0x08CC	CANB_RXIMR19—Rx individual mask register 19	R/W	0x0000_0000	29.3.4.11/29-26
0x08D0	CANB_RXIMR20—Rx individual mask register 20	R/W	0x0000_0000	29.3.4.11/29-26
0x08D4	CANB_RXIMR21—Rx individual mask register 21	R/W	0x0000_0000	29.3.4.11/29-26
0x08D8	CANB_RXIMR22—Rx individual mask register 22	R/W	0x0000_0000	29.3.4.11/29-26
0x08DC	CANB_RXIMR23—Rx individual mask register 23	R/W	0x0000_0000	29.3.4.11/29-26
0x08E0	CANB_RXIMR24—Rx individual mask register 24	R/W	0x0000_0000	29.3.4.11/29-26
0x08E4	CANB_RXIMR25—Rx individual mask register 25	R/W	0x0000_0000	29.3.4.11/29-26
0x08E8	CANB_RXIMR26—Rx individual mask register 26	R/W	0x0000_0000	29.3.4.11/29-26

Table A-4. PXN20 Detailed Register Map (continued)

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
0x08EC	CANB_RXIMR27—Rx individual mask register 27	R/W	0x0000_0000	29.3.4.11/29-26
0x08F0	CANB_RXIMR28—Rx individual mask register 28	R/W	0x0000_0000	29.3.4.11/29-26
0x08F4	CANB_RXIMR29—Rx individual mask register 29	R/W	0x0000_0000	29.3.4.11/29-26
0x08F8	CANB_RXIMR30—Rx individual mask register 30	R/W	0x0000_0000	29.3.4.11/29-26
0x08FC	CANB_RXIMR31—Rx individual mask register 31	R/W	0x0000_0000	29.3.4.11/29-26
0x0900	CANB_RXIMR32—Rx individual mask register 32	R/W	0x0000_0000	29.3.4.11/29-26
0x0904	CANB_RXIMR33—Rx individual mask register 33	R/W	0x0000_0000	29.3.4.11/29-26
0x0908	CANB_RXIMR34—Rx individual mask register 34	R/W	0x0000_0000	29.3.4.11/29-26
0x090C	CANB_RXIMR35—Rx individual mask register 35	R/W	0x0000_0000	29.3.4.11/29-26
0x0910	CANB_RXIMR36—Rx individual mask register 36	R/W	0x0000_0000	29.3.4.11/29-26
0x0914	CANB_RXIMR37—Rx individual mask register 37	R/W	0x0000_0000	29.3.4.11/29-26
0x0918	CANB_RXIMR38—Rx individual mask register 38	R/W	0x0000_0000	29.3.4.11/29-26
0x091C	CANB_RXIMR39—Rx individual mask register 39	R/W	0x0000_0000	29.3.4.11/29-26
0x0920	CANB_RXIMR40—Rx individual mask register 40	R/W	0x0000_0000	29.3.4.11/29-26
0x0924	CANB_RXIMR41—Rx individual mask register 41	R/W	0x0000_0000	29.3.4.11/29-26
0x0928	CANB_RXIMR42—Rx individual mask register 42	R/W	0x0000_0000	29.3.4.11/29-26
0x092C	CANB_RXIMR43—Rx individual mask register 43	R/W	0x0000_0000	29.3.4.11/29-26
0x0930	CANB_RXIMR44—Rx individual mask register 44	R/W	0x0000_0000	29.3.4.11/29-26
0x0934	CANB_RXIMR45—Rx individual mask register 45	R/W	0x0000_0000	29.3.4.11/29-26
0x0938	CANB_RXIMR46—Rx individual mask register 46	R/W	0x0000_0000	29.3.4.11/29-26
0x093C	CANB_RXIMR47—Rx individual mask register 47	R/W	0x0000_0000	29.3.4.11/29-26
0x0940	CANB_RXIMR48—Rx individual mask register 48	R/W	0x0000_0000	29.3.4.11/29-26
0x0944	CANB_RXIMR49—Rx individual mask register 49	R/W	0x0000_0000	29.3.4.11/29-26
0x0948	CANB_RXIMR50—Rx individual mask register 50	R/W	0x0000_0000	29.3.4.11/29-26
0x094C	CANB_RXIMR51—Rx individual mask register 51	R/W	0x0000_0000	29.3.4.11/29-26
0x0950	CANB_RXIMR52—Rx individual mask register 52	R/W	0x0000_0000	29.3.4.11/29-26
0x0954	CANB_RXIMR53—Rx individual mask register 53	R/W	0x0000_0000	29.3.4.11/29-26
0x0958	CANB_RXIMR54—Rx individual mask register 54	R/W	0x0000_0000	29.3.4.11/29-26
0x095C	CANB_RXIMR55—Rx individual mask register 55	R/W	0x0000_0000	29.3.4.11/29-26
0x0960	CANB_RXIMR56—Rx individual mask register 56	R/W	0x0000_0000	29.3.4.11/29-26
0x0964	CANB_RXIMR57—Rx individual mask register 57	R/W	0x0000_0000	29.3.4.11/29-26
0x0968	CANB_RXIMR58—Rx individual mask register 58	R/W	0x0000_0000	29.3.4.11/29-26

Table A-4. PXN20 Detailed Register Map (continued)

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
0x096C	CANB_RXIMR59—Rx individual mask register 59	R/W	0x0000_0000	29.3.4.11/29-26
0x0970	CANB_RXIMR60—Rx individual mask register 60	R/W	0x0000_0000	29.3.4.11/29-26
0x0974	CANB_RXIMR61—Rx individual mask register 61	R/W	0x0000_0000	29.3.4.11/29-26
0x0978	CANB_RXIMR62—Rx individual mask register 62	R/W	0x0000_0000	29.3.4.11/29-26
0x097C	CANB_RXIMR63—Rx individual mask register 63	R/W	0x0000_0000	29.3.4.11/29-26
0x0980–0x3FFF	Reserved			
0xFFFC_8000	**FlexCAN_C** **Chapter 29, Controller Area Network (FlexCAN)**			
0x0000	CANC_MCR—Module configuration register	R/W	0x9890_000F	29.3.4.1/29-11
0x0004	CANC_CTRL—control register	R/W	0x0000_0000	29.3.4.2/29-14
0x0008	CANC_TIMER—Free-running timer	R/W	0x0000_0000	29.3.4.3/29-17
0x000C	Reserved			
0x0010	CANC_RXGMASK—Rx global mask	R/W	0xFFFF_FFFF	29.3.4.4.1/29-18
0x0014	CANC_RX14MASK—Rx buffer 14 mask	R/W	0xFFFF_FFFF	29.3.4.4.2/29-19
0x0018	CANC_RX15MASK—Rx buffer 15 mask	R/W	0xFFFF_FFFF	29.3.4.4.3/29-19
0x001C	CANC_ECR—Error counter register	R/W	0x0000_0000	29.3.2/29-7
0x0020	CANC_ESR—Error and status register	R/W	0x0000_0000	29.3.4.6/29-21
0x0024	CANC_IMASK2—Interrupt masks 2	R/W	0x0000_0000	29.3.4.7/29-23
0x0028	CANC_IMASK1—Interrupt masks 1	R/W	0x0000_0000	29.3.4.8/29-24
0x002C	CANC_IFLAG2—Interrupt flags 2	R/W	0x0000_0000	29.3.4.9/29-24
0x0030	CANC_IFLAG1—Interrupt flags 1	R/W	0x0000_0000	29.3.4.10/29-25
0x0034–0x007F	Reserved			
0x0080	MB0—Message buffer 0	R/W	0x0000_0000	29.3.2/29-7
0x0090	MB1—Message buffer 1	R/W	0x0000_0000	29.3.2/29-7
0x00AO	MB2—Message buffer 2	R/W	0x0000_0000	29.3.2/29-7
0x00B0	MB3—Message buffer 3	R/W	0x0000_0000	29.3.2/29-7
0x00C0	MB4—Message buffer 4	R/W	0x0000_0000	29.3.2/29-7
0x00D0	MB5—Message buffer 5	R/W	0x0000_0000	29.3.2/29-7
0x00E0	MB6—Message buffer 6	R/W	0x0000_0000	29.3.2/29-7
0x00F0	MB7—Message buffer 7	R/W	0x0000_0000	29.3.2/29-7

Table A-4. PXN20 Detailed Register Map (continued)

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
0x0100	MB8—Message buffer 8	R/W	0x0000_0000	29.3.2/29-7
0x0110	MB9—Message buffer 9	R/W	0x0000_0000	29.3.2/29-7
0x0120	MB10—Message buffer 10	R/W	0x0000_0000	29.3.2/29-7
0x0130	MB11—Message buffer 11	R/W	0x0000_0000	29.3.2/29-7
0x0140	MB12—Message buffer 12	R/W	0x0000_0000	29.3.2/29-7
0x0150	MB13—Message buffer 13	R/W	0x0000_0000	29.3.2/29-7
0x0160	MB14—Message buffer 14	R/W	0x0000_0000	29.3.2/29-7
0x0170	MB15—Message buffer 15	R/W	0x0000_0000	29.3.2/29-7
0x0180	MB16—Message buffer 16	R/W	0x0000_0000	29.3.2/29-7
0x0190	MB17—Message buffer 17	R/W	0x0000_0000	29.3.2/29-7
0x01A0	MB18—Message buffer 18	R/W	0x0000_0000	29.3.2/29-7
0x01B0	MB19—Message buffer 19	R/W	0x0000_0000	29.3.2/29-7
0x01C0	MB20—Message buffer 20	R/W	0x0000_0000	29.3.2/29-7
0x01D0	MB21—Message buffer 21	R/W	0x0000_0000	29.3.2/29-7
0x01E0	MB22—Message buffer 22	R/W	0x0000_0000	29.3.2/29-7
0x01F0	MB23—Message buffer 23	R/W	0x0000_0000	29.3.2/29-7
0x0200	MB24—Message buffer 24	R/W	0x0000_0000	29.3.2/29-7
0x0210	MB25—Message buffer 25	R/W	0x0000_0000	29.3.2/29-7
0x0220	MB26—Message buffer 26	R/W	0x0000_0000	29.3.2/29-7
0x0230	MB27—Message buffer 27	R/W	0x0000_0000	29.3.2/29-7
0x0240	MB28—Message buffer 28	R/W	0x0000_0000	29.3.2/29-7
0x0250	MB29—Message buffer 29	R/W	0x0000_0000	29.3.2/29-7
0x0260	MB30—Message buffer 30	R/W	0x0000_0000	29.3.2/29-7
0x0270	MB31—Message buffer 31	R/W	0x0000_0000	29.3.2/29-7
0x0280	MB32—Message buffer 32	R/W	0x0000_0000	29.3.2/29-7
0x0290	MB33—Message buffer 33	R/W	0x0000_0000	29.3.2/29-7
0x02A0	MB34—Message buffer 34	R/W	0x0000_0000	29.3.2/29-7
0x02B0	MB35—Message buffer 35	R/W	0x0000_0000	29.3.2/29-7
0x02C0	MB36—Message buffer 36	R/W	0x0000_0000	29.3.2/29-7
0x02D0	MB37—Message buffer 37	R/W	0x0000_0000	29.3.2/29-7
0x02E0	MB38—Message buffer 38	R/W	0x0000_0000	29.3.2/29-7
0x02F0	MB39—Message buffer 39	R/W	0x0000_0000	29.3.2/29-7

Table A-4. PXN20 Detailed Register Map (continued)

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
0x0300	MB40—Message buffer 40	R/W	0x0000_0000	29.3.2/29-7
0x0310	MB41—Message buffer 41	R/W	0x0000_0000	29.3.2/29-7
0x0320	MB42—Message buffer 42	R/W	0x0000_0000	29.3.2/29-7
0x0330	MB43—Message buffer 43	R/W	0x0000_0000	29.3.2/29-7
0x0340	MB44—Message buffer 44	R/W	0x0000_0000	29.3.2/29-7
0x0350	MB45—Message buffer 45	R/W	0x0000_0000	29.3.2/29-7
0x0360	MB46—Message buffer 46	R/W	0x0000_0000	29.3.2/29-7
0x0370	MB47—Message buffer 47	R/W	0x0000_0000	29.3.2/29-7
0x0380	MB48—Message buffer 48	R/W	0x0000_0000	29.3.2/29-7
0x0390	MB49—Message buffer 49	R/W	0x0000_0000	29.3.2/29-7
0x03A0	MB50—Message buffer 50	R/W	0x0000_0000	29.3.2/29-7
0x03B0	MB51—Message buffer 51	R/W	0x0000_0000	29.3.2/29-7
0x03C0	MB52—Message buffer 52	R/W	0x0000_0000	29.3.2/29-7
0x03D0	MB53—Message buffer 53	R/W	0x0000_0000	29.3.2/29-7
0x03E0	MB54—Message buffer 54	R/W	0x0000_0000	29.3.2/29-7
0x03F0	MB55—Message buffer 55	R/W	0x0000_0000	29.3.2/29-7
0x0400	MB56—Message buffer 56	R/W	0x0000_0000	29.3.2/29-7
0x0410	MB57—Message buffer 57	R/W	0x0000_0000	29.3.2/29-7
0x0420	MB58—Message buffer 58	R/W	0x0000_0000	29.3.2/29-7
0x0430	MB59—Message buffer 59	R/W	0x0000_0000	29.3.2/29-7
0x0440	MB60—Message buffer 60	R/W	0x0000_0000	29.3.2/29-7
0x0450	MB61—Message buffer 61	R/W	0x0000_0000	29.3.2/29-7
0x0460	MB62—Message buffer 62	R/W	0x0000_0000	29.3.2/29-7
0x0470	MB63—Message buffer 63	R/W	0x0000_0000	29.3.2/29-7
0x0480–087F	Reserved			
0x0880	CANC_RXIMR0—Rx individual mask register 0	R/W	0x0000_0000	29.3.4.11/29-26
0x0884	CANC_RXIMR1—Rx individual mask register 1	R/W	0x0000_0000	29.3.4.11/29-26
0x0888	CANC_RXIMR2—Rx individual mask register 2	R/W	0x0000_0000	29.3.4.11/29-26
0x088C	CANC_RXIMR3—Rx individual mask register 3	R/W	0x0000_0000	29.3.4.11/29-26
0x0890	CANC_RXIMR4—Rx individual mask register 4	R/W	0x0000_0000	29.3.4.11/29-26
0x0894	CANC_RXIMR5—Rx individual mask register 5	R/W	0x0000_0000	29.3.4.11/29-26
0x0898	CANC_RXIMR6—Rx individual mask register 6	R/W	0x0000_0000	29.3.4.11/29-26

Table A-4. PXN20 Detailed Register Map (continued)

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
0x089C	CANC_RXIMR7—Rx individual mask register 7	R/W	0x0000_0000	29.3.4.11/29-26
0x08A0	CANC_RXIMR8—Rx individual mask register 8	R/W	0x0000_0000	29.3.4.11/29-26
0x08A4	CANC_RXIMR9—Rx individual mask register 9	R/W	0x0000_0000	29.3.4.11/29-26
0x08A8	CANC_RXIMR10—Rx individual mask register 10	R/W	0x0000_0000	29.3.4.11/29-26
0x08AC	CANC_RXIMR11—Rx individual mask register 11	R/W	0x0000_0000	29.3.4.11/29-26
0x08B0	CANC_RXIMR12—Rx individual mask register 12	R/W	0x0000_0000	29.3.4.11/29-26
0x08B4	CANC_RXIMR13—Rx individual mask register 13	R/W	0x0000_0000	29.3.4.11/29-26
0x08B8	CANC_RXIMR14—Rx individual mask register 14	R/W	0x0000_0000	29.3.4.11/29-26
0x08BC	CANC_RXIMR15—Rx individual mask register 15	R/W	0x0000_0000	29.3.4.11/29-26
0x08C0	CANC_RXIMR16—Rx individual mask register 16	R/W	0x0000_0000	29.3.4.11/29-26
0x08C4	CANC_RXIMR17—Rx individual mask register 17	R/W	0x0000_0000	29.3.4.11/29-26
0x08C8	CANC_RXIMR18—Rx individual mask register 18	R/W	0x0000_0000	29.3.4.11/29-26
0x08CC	CANC_RXIMR19—Rx individual mask register 19	R/W	0x0000_0000	29.3.4.11/29-26
0x08D0	CANC_RXIMR20—Rx individual mask register 20	R/W	0x0000_0000	29.3.4.11/29-26
0x08D4	CANC_RXIMR21—Rx individual mask register 21	R/W	0x0000_0000	29.3.4.11/29-26
0x08D8	CANC_RXIMR22—Rx individual mask register 22	R/W	0x0000_0000	29.3.4.11/29-26
0x08DC	CANC_RXIMR23—Rx individual mask register 23	R/W	0x0000_0000	29.3.4.11/29-26
0x08E0	CANC_RXIMR24—Rx individual mask register 24	R/W	0x0000_0000	29.3.4.11/29-26
0x08E4	CANC_RXIMR25—Rx individual mask register 25	R/W	0x0000_0000	29.3.4.11/29-26
0x08E8	CANC_RXIMR26—Rx individual mask register 26	R/W	0x0000_0000	29.3.4.11/29-26
0x08EC	CANC_RXIMR27—Rx individual mask register 27	R/W	0x0000_0000	29.3.4.11/29-26
0x08F0	CANC_RXIMR28—Rx individual mask register 28	R/W	0x0000_0000	29.3.4.11/29-26
0x08F4	CANC_RXIMR29—Rx individual mask register 29	R/W	0x0000_0000	29.3.4.11/29-26
0x08F8	CANC_RXIMR30—Rx individual mask register 30	R/W	0x0000_0000	29.3.4.11/29-26
0x08FC	CANC_RXIMR31—Rx individual mask register 31	R/W	0x0000_0000	29.3.4.11/29-26
0x0900	CANC_RXIMR32—Rx individual mask register 32	R/W	0x0000_0000	29.3.4.11/29-26
0x0904	CANC_RXIMR33—Rx individual mask register 33	R/W	0x0000_0000	29.3.4.11/29-26
0x0908	CANC_RXIMR34—Rx individual mask register 34	R/W	0x0000_0000	29.3.4.11/29-26
0x090C	CANC_RXIMR35—Rx individual mask register 35	R/W	0x0000_0000	29.3.4.11/29-26
0x0910	CANC_RXIMR36—Rx individual mask register 36	R/W	0x0000_0000	29.3.4.11/29-26
0x0914	CANC_RXIMR37—Rx individual mask register 37	R/W	0x0000_0000	29.3.4.11/29-26
0x0918	CANC_RXIMR38—Rx individual mask register 38	R/W	0x0000_0000	29.3.4.11/29-26

Table A-4. PXN20 Detailed Register Map (continued)

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
0x091C	CANC_RXIMR39—Rx individual mask register 39	R/W	0x0000_0000	29.3.4.11/29-26
0x0920	CANC_RXIMR40—Rx individual mask register 40	R/W	0x0000_0000	29.3.4.11/29-26
0x0924	CANC_RXIMR41—Rx individual mask register 41	R/W	0x0000_0000	29.3.4.11/29-26
0x0928	CANC_RXIMR42—Rx individual mask register 42	R/W	0x0000_0000	29.3.4.11/29-26
0x092C	CANC_RXIMR43—Rx individual mask register 43	R/W	0x0000_0000	29.3.4.11/29-26
0x0930	CANC_RXIMR44—Rx individual mask register 44	R/W	0x0000_0000	29.3.4.11/29-26
0x0934	CANC_RXIMR45—Rx individual mask register 45	R/W	0x0000_0000	29.3.4.11/29-26
0x0938	CANC_RXIMR46—Rx individual mask register 46	R/W	0x0000_0000	29.3.4.11/29-26
0x093C	CANC_RXIMR47—Rx individual mask register 47	R/W	0x0000_0000	29.3.4.11/29-26
0x0940	CANC_RXIMR48—Rx individual mask register 48	R/W	0x0000_0000	29.3.4.11/29-26
0x0944	CANC_RXIMR49—Rx individual mask register 49	R/W	0x0000_0000	29.3.4.11/29-26
0x0948	CANC_RXIMR50—Rx individual mask register 50	R/W	0x0000_0000	29.3.4.11/29-26
0x094C	CANC_RXIMR51—Rx individual mask register 51	R/W	0x0000_0000	29.3.4.11/29-26
0x0950	CANC_RXIMR52—Rx individual mask register 52	R/W	0x0000_0000	29.3.4.11/29-26
0x0954	CANC_RXIMR53—Rx individual mask register 53	R/W	0x0000_0000	29.3.4.11/29-26
0x0958	CANC_RXIMR54—Rx individual mask register 54	R/W	0x0000_0000	29.3.4.11/29-26
0x095C	CANC_RXIMR55—Rx individual mask register 55	R/W	0x0000_0000	29.3.4.11/29-26
0x0960	CANC_RXIMR56—Rx individual mask register 56	R/W	0x0000_0000	29.3.4.11/29-26
0x0964	CANC_RXIMR57—Rx individual mask register 57	R/W	0x0000_0000	29.3.4.11/29-26
0x0968	CANC_RXIMR58—Rx individual mask register 58	R/W	0x0000_0000	29.3.4.11/29-26
0x096C	CANC_RXIMR59—Rx individual mask register 59	R/W	0x0000_0000	29.3.4.11/29-26
0x0970	CANC_RXIMR60—Rx individual mask register 60	R/W	0x0000_0000	29.3.4.11/29-26
0x0974	CANC_RXIMR61—Rx individual mask register 61	R/W	0x0000_0000	29.3.4.11/29-26
0x0978	CANC_RXIMR62—Rx individual mask register 62	R/W	0x0000_0000	29.3.4.11/29-26
0x097C	CANC_RXIMR63—Rx individual mask register 63	R/W	0x0000_0000	29.3.4.11/29-26
0x0980–0x3FFF	Reserved			
0xFFFC_C000	**FlexCAN_D** **Chapter 29, Controller Area Network (FlexCAN)**			
0x0000	CAND_MCR—Module configuration register	R/W	0x9890_000F	29.3.4.1/29-11
0x0004	CAND_CTRL—control register	R/W	0x0000_0000	29.3.4.2/29-14
0x0008	CAND_TIMER—Free-running timer	R/W	0x0000_0000	29.3.4.3/29-17
0x000C	Reserved			

Table A-4. PXN20 Detailed Register Map (continued)

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
0x0010	CAND_RXGMASK—Rx global mask	R/W	0xFFFF_FFFF	29.3.4.4.1/29-18
0x0014	CAND_RX14MASK—Rx buffer 14 mask	R/W	0xFFFF_FFFF	29.3.4.4.2/29-19
0x0018	CAND_RX15MASK—Rx buffer 15 mask	R/W	0xFFFF_FFFF	29.3.4.4.3/29-19
0x001C	CAND_ECR—Error counter register	R/W	0x0000_0000	29.3.2/29-7
0x0020	CAND_ESR—Error and status register	R/W	0x0000_0000	29.3.4.6/29-21
0x0024	CAND_IMASK2—Interrupt masks 2	R/W	0x0000_0000	29.3.4.7/29-23
0x0028	CAND_IMASK1—Interrupt masks 1	R/W	0x0000_0000	29.3.4.8/29-24
0x002C	CAND_IFLAG2—Interrupt flags 2	R/W	0x0000_0000	29.3.4.9/29-24
0x0030	CAND_IFLAG1—Interrupt flags 1	R/W	0x0000_0000	29.3.4.10/29-25
0x0034–0x007F	Reserved			
0x0080	MB0—Message buffer 0	R/W	0x0000_0000	29.3.2/29-7
0x0090	MB1—Message buffer 1	R/W	0x0000_0000	29.3.2/29-7
0x00AO	MB2—Message buffer 2	R/W	0x0000_0000	29.3.2/29-7
0x00B0	MB3—Message buffer 3	R/W	0x0000_0000	29.3.2/29-7
0x00C0	MB4—Message buffer 4	R/W	0x0000_0000	29.3.2/29-7
0x00D0	MB5—Message buffer 5	R/W	0x0000_0000	29.3.2/29-7
0x00E0	MB6—Message buffer 6	R/W	0x0000_0000	29.3.2/29-7
0x00F0	MB7—Message buffer 7	R/W	0x0000_0000	29.3.2/29-7
0x0100	MB8—Message buffer 8	R/W	0x0000_0000	29.3.2/29-7
0x0110	MB9—Message buffer 9	R/W	0x0000_0000	29.3.2/29-7
0x0120	MB10—Message buffer 10	R/W	0x0000_0000	29.3.2/29-7
0x0130	MB11—Message buffer 11	R/W	0x0000_0000	29.3.2/29-7
0x0140	MB12—Message buffer 12	R/W	0x0000_0000	29.3.2/29-7
0x0150	MB13—Message buffer 13	R/W	0x0000_0000	29.3.2/29-7
0x0160	MB14—Message buffer 14	R/W	0x0000_0000	29.3.2/29-7
0x0170	MB15—Message buffer 15	R/W	0x0000_0000	29.3.2/29-7
0x0180	MB16—Message buffer 16	R/W	0x0000_0000	29.3.2/29-7
0x0190	MB17—Message buffer 17	R/W	0x0000_0000	29.3.2/29-7
0x01A0	MB18—Message buffer 18	R/W	0x0000_0000	29.3.2/29-7
0x01B0	MB19—Message buffer 19	R/W	0x0000_0000	29.3.2/29-7

Table A-4. PXN20 Detailed Register Map (continued)

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
0x01C0	MB20—Message buffer 20	R/W	0x0000_0000	29.3.2/29-7
0x01D0	MB21—Message buffer 21	R/W	0x0000_0000	29.3.2/29-7
0x01E0	MB22—Message buffer 22	R/W	0x0000_0000	29.3.2/29-7
0x01F0	MB23—Message buffer 23	R/W	0x0000_0000	29.3.2/29-7
0x0200	MB24—Message buffer 24	R/W	0x0000_0000	29.3.2/29-7
0x0210	MB25—Message buffer 25	R/W	0x0000_0000	29.3.2/29-7
0x0220	MB26—Message buffer 26	R/W	0x0000_0000	29.3.2/29-7
0x0230	MB27—Message buffer 27	R/W	0x0000_0000	29.3.2/29-7
0x0240	MB28—Message buffer 28	R/W	0x0000_0000	29.3.2/29-7
0x0250	MB29—Message buffer 29	R/W	0x0000_0000	29.3.2/29-7
0x0260	MB30—Message buffer 30	R/W	0x0000_0000	29.3.2/29-7
0x0270	MB31—Message buffer 31	R/W	0x0000_0000	29.3.2/29-7
0x0280	MB32—Message buffer 32	R/W	0x0000_0000	29.3.2/29-7
0x0290	MB33—Message buffer 33	R/W	0x0000_0000	29.3.2/29-7
0x02A0	MB34—Message buffer 34	R/W	0x0000_0000	29.3.2/29-7
0x02B0	MB35—Message buffer 35	R/W	0x0000_0000	29.3.2/29-7
0x02C0	MB36—Message buffer 36	R/W	0x0000_0000	29.3.2/29-7
0x02D0	MB37—Message buffer 37	R/W	0x0000_0000	29.3.2/29-7
0x02E0	MB38—Message buffer 38	R/W	0x0000_0000	29.3.2/29-7
0x02F0	MB39—Message buffer 39	R/W	0x0000_0000	29.3.2/29-7
0x0300	MB40—Message buffer 40	R/W	0x0000_0000	29.3.2/29-7
0x0310	MB41—Message buffer 41	R/W	0x0000_0000	29.3.2/29-7
0x0320	MB42—Message buffer 42	R/W	0x0000_0000	29.3.2/29-7
0x0330	MB43—Message buffer 43	R/W	0x0000_0000	29.3.2/29-7
0x0340	MB44—Message buffer 44	R/W	0x0000_0000	29.3.2/29-7
0x0350	MB45—Message buffer 45	R/W	0x0000_0000	29.3.2/29-7
0x0360	MB46—Message buffer 46	R/W	0x0000_0000	29.3.2/29-7
0x0370	MB47—Message buffer 47	R/W	0x0000_0000	29.3.2/29-7
0x0380	MB48—Message buffer 48	R/W	0x0000_0000	29.3.2/29-7
0x0390	MB49—Message buffer 49	R/W	0x0000_0000	29.3.2/29-7
0x03A0	MB50—Message buffer 50	R/W	0x0000_0000	29.3.2/29-7
0x03B0	MB51—Message buffer 51	R/W	0x0000_0000	29.3.2/29-7

Table A-4. PXN20 Detailed Register Map (continued)

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
0x03C0	MB52—Message buffer 52	R/W	0x0000_0000	29.3.2/29-7
0x03D0	MB53—Message buffer 53	R/W	0x0000_0000	29.3.2/29-7
0x03E0	MB54—Message buffer 54	R/W	0x0000_0000	29.3.2/29-7
0x03F0	MB55—Message buffer 55	R/W	0x0000_0000	29.3.2/29-7
0x0400	MB56—Message buffer 56	R/W	0x0000_0000	29.3.2/29-7
0x0410	MB57—Message buffer 57	R/W	0x0000_0000	29.3.2/29-7
0x0420	MB58—Message buffer 58	R/W	0x0000_0000	29.3.2/29-7
0x0430	MB59—Message buffer 59	R/W	0x0000_0000	29.3.2/29-7
0x0440	MB60—Message buffer 60	R/W	0x0000_0000	29.3.2/29-7
0x0450	MB61—Message buffer 61	R/W	0x0000_0000	29.3.2/29-7
0x0460	MB62—Message buffer 62	R/W	0x0000_0000	29.3.2/29-7
0x0470	MB63—Message buffer 63	R/W	0x0000_0000	29.3.2/29-7
0x0480–087F	Reserved			
0x0880	CAND_RXIMR0—Rx individual mask register 0	R/W	0x0000_0000	29.3.4.11/29-26
0x0884	CAND_RXIMR1—Rx individual mask register 1	R/W	0x0000_0000	29.3.4.11/29-26
0x0888	CAND_RXIMR2—Rx individual mask register 2	R/W	0x0000_0000	29.3.4.11/29-26
0x088C	CAND_RXIMR3—Rx individual mask register 3	R/W	0x0000_0000	29.3.4.11/29-26
0x0890	CAND_RXIMR4—Rx individual mask register 4	R/W	0x0000_0000	29.3.4.11/29-26
0x0894	CAND_RXIMR5—Rx individual mask register 5	R/W	0x0000_0000	29.3.4.11/29-26
0x0898	CAND_RXIMR6—Rx individual mask register 6	R/W	0x0000_0000	29.3.4.11/29-26
0x089C	CAND_RXIMR7—Rx individual mask register 7	R/W	0x0000_0000	29.3.4.11/29-26
0x08A0	CAND_RXIMR8—Rx individual mask register 8	R/W	0x0000_0000	29.3.4.11/29-26
0x08A4	CAND_RXIMR9—Rx individual mask register 9	R/W	0x0000_0000	29.3.4.11/29-26
0x08A8	CAND_RXIMR10—Rx individual mask register 10	R/W	0x0000_0000	29.3.4.11/29-26
0x08AC	CAND_RXIMR11—Rx individual mask register 11	R/W	0x0000_0000	29.3.4.11/29-26
0x08B0	CAND_RXIMR12—Rx individual mask register 12	R/W	0x0000_0000	29.3.4.11/29-26
0x08B4	CAND_RXIMR13—Rx individual mask register 13	R/W	0x0000_0000	29.3.4.11/29-26
0x08B8	CAND_RXIMR14—Rx individual mask register 14	R/W	0x0000_0000	29.3.4.11/29-26
0x08BC	CAND_RXIMR15—Rx individual mask register 15	R/W	0x0000_0000	29.3.4.11/29-26
0x08C0	CAND_RXIMR16—Rx individual mask register 16	R/W	0x0000_0000	29.3.4.11/29-26
0x08C4	CAND_RXIMR17—Rx individual mask register 17	R/W	0x0000_0000	29.3.4.11/29-26
0x08C8	CAND_RXIMR18—Rx individual mask register 18	R/W	0x0000_0000	29.3.4.11/29-26

Table A-4. PXN20 Detailed Register Map (continued)

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
0x08CC	CAND_RXIMR19—Rx individual mask register 19	R/W	0x0000_0000	29.3.4.11/29-26
0x08D0	CAND_RXIMR20—Rx individual mask register 20	R/W	0x0000_0000	29.3.4.11/29-26
0x08D4	CAND_RXIMR21—Rx individual mask register 21	R/W	0x0000_0000	29.3.4.11/29-26
0x08D8	CAND_RXIMR22—Rx individual mask register 22	R/W	0x0000_0000	29.3.4.11/29-26
0x08DC	CAND_RXIMR23—Rx individual mask register 23	R/W	0x0000_0000	29.3.4.11/29-26
0x08E0	CAND_RXIMR24—Rx individual mask register 24	R/W	0x0000_0000	29.3.4.11/29-26
0x08E4	CAND_RXIMR25—Rx individual mask register 25	R/W	0x0000_0000	29.3.4.11/29-26
0x08E8	CAND_RXIMR26—Rx individual mask register 26	R/W	0x0000_0000	29.3.4.11/29-26
0x08EC	CAND_RXIMR27—Rx individual mask register 27	R/W	0x0000_0000	29.3.4.11/29-26
0x08F0	CAND_RXIMR28—Rx individual mask register 28	R/W	0x0000_0000	29.3.4.11/29-26
0x08F4	CAND_RXIMR29—Rx individual mask register 29	R/W	0x0000_0000	29.3.4.11/29-26
0x08F8	CAND_RXIMR30—Rx individual mask register 30	R/W	0x0000_0000	29.3.4.11/29-26
0x08FC	CAND_RXIMR31—Rx individual mask register 31	R/W	0x0000_0000	29.3.4.11/29-26
0x0900	CAND_RXIMR32—Rx individual mask register 32	R/W	0x0000_0000	29.3.4.11/29-26
0x0904	CAND_RXIMR33—Rx individual mask register 33	R/W	0x0000_0000	29.3.4.11/29-26
0x0908	CAND_RXIMR34—Rx individual mask register 34	R/W	0x0000_0000	29.3.4.11/29-26
0x090C	CAND_RXIMR35—Rx individual mask register 35	R/W	0x0000_0000	29.3.4.11/29-26
0x0910	CAND_RXIMR36—Rx individual mask register 36	R/W	0x0000_0000	29.3.4.11/29-26
0x0914	CAND_RXIMR37—Rx individual mask register 37	R/W	0x0000_0000	29.3.4.11/29-26
0x0918	CAND_RXIMR38—Rx individual mask register 38	R/W	0x0000_0000	29.3.4.11/29-26
0x091C	CAND_RXIMR39—Rx individual mask register 39	R/W	0x0000_0000	29.3.4.11/29-26
0x0920	CAND_RXIMR40—Rx individual mask register 40	R/W	0x0000_0000	29.3.4.11/29-26
0x0924	CAND_RXIMR41—Rx individual mask register 41	R/W	0x0000_0000	29.3.4.11/29-26
0x0928	CAND_RXIMR42—Rx individual mask register 42	R/W	0x0000_0000	29.3.4.11/29-26
0x092C	CAND_RXIMR43—Rx individual mask register 43	R/W	0x0000_0000	29.3.4.11/29-26
0x0930	CAND_RXIMR44—Rx individual mask register 44	R/W	0x0000_0000	29.3.4.11/29-26
0x0934	CAND_RXIMR45—Rx individual mask register 45	R/W	0x0000_0000	29.3.4.11/29-26
0x0938	CAND_RXIMR46—Rx individual mask register 46	R/W	0x0000_0000	29.3.4.11/29-26
0x093C	CAND_RXIMR47—Rx individual mask register 47	R/W	0x0000_0000	29.3.4.11/29-26
0x0940	CAND_RXIMR48—Rx individual mask register 48	R/W	0x0000_0000	29.3.4.11/29-26
0x0944	CAND_RXIMR49—Rx individual mask register 49	R/W	0x0000_0000	29.3.4.11/29-26
0x0948	CAND_RXIMR50—Rx individual mask register 50	R/W	0x0000_0000	29.3.4.11/29-26

Table A-4. PXN20 Detailed Register Map (continued)

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
0x094C	CAND_RXIMR51—Rx individual mask register 51	R/W	0x0000_0000	29.3.4.11/29-26
0x0950	CAND_RXIMR52—Rx individual mask register 52	R/W	0x0000_0000	29.3.4.11/29-26
0x0954	CAND_RXIMR53—Rx individual mask register 53	R/W	0x0000_0000	29.3.4.11/29-26
0x0958	CAND_RXIMR54—Rx individual mask register 54	R/W	0x0000_0000	29.3.4.11/29-26
0x095C	CAND_RXIMR55—Rx individual mask register 55	R/W	0x0000_0000	29.3.4.11/29-26
0x0960	CAND_RXIMR56—Rx individual mask register 56	R/W	0x0000_0000	29.3.4.11/29-26
0x0964	CAND_RXIMR57—Rx individual mask register 57	R/W	0x0000_0000	29.3.4.11/29-26
0x0968	CAND_RXIMR58—Rx individual mask register 58	R/W	0x0000_0000	29.3.4.11/29-26
0x096C	CAND_RXIMR59—Rx individual mask register 59	R/W	0x0000_0000	29.3.4.11/29-26
0x0970	CAND_RXIMR60—Rx individual mask register 60	R/W	0x0000_0000	29.3.4.11/29-26
0x0974	CAND_RXIMR61—Rx individual mask register 61	R/W	0x0000_0000	29.3.4.11/29-26
0x0978	CAND_RXIMR62—Rx individual mask register 62	R/W	0x0000_0000	29.3.4.11/29-26
0x097C	CAND_RXIMR63—Rx individual mask register 63	R/W	0x0000_0000	29.3.4.11/29-26
0x0980–0x3FFF	Reserved			
0xFFFD_0000	**FlexCAN_E** **Chapter 29, Controller Area Network (FlexCAN)**			
0x0000	CANE_MCR—Module configuration register	R/W	0x9890_000F	29.3.4.1/29-11
0x0004	CANE_CTRL—control register	R/W	0x0000_0000	29.3.4.2/29-14
0x0008	CANE_TIMER—Free-running timer	R/W	0x0000_0000	29.3.4.3/29-17
0x000C	Reserved			
0x0010	CANE_RXGMASK—Rx global mask	R/W	0xFFFF_FFFF	29.3.4.4.1/29-18
0x0014	CANE_RX14MASK—Rx buffer 14 mask	R/W	0xFFFF_FFFF	29.3.4.4.2/29-19
0x0018	CANE_RX15MASK—Rx buffer 15 mask	R/W	0xFFFF_FFFF	29.3.4.4.3/29-19
0x001C	CANE_ECR—Error counter register	R/W	0x0000_0000	29.3.2/29-7
0x0020	CANE_ESR—Error and status register	R/W	0x0000_0000	29.3.4.6/29-21
0x0024	CANE_IMASK2—Interrupt masks 2	R/W	0x0000_0000	29.3.4.7/29-23
0x0028	CANE_IMASK1—Interrupt masks 1	R/W	0x0000_0000	29.3.4.8/29-24
0x002C	CANE_IFLAG2—Interrupt flags 2	R/W	0x0000_0000	29.3.4.9/29-24
0x0030	CANE_IFLAG1—Interrupt flags 1	R/W	0x0000_0000	29.3.4.10/29-25
0x0034–0x007F	Reserved			

Table A-4. PXN20 Detailed Register Map (continued)

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
0x0080	MB0—Message buffer 0	R/W	0x0000_0000	29.3.2/29-7
0x0090	MB1—Message buffer 1	R/W	0x0000_0000	29.3.2/29-7
0x00AO	MB2—Message buffer 2	R/W	0x0000_0000	29.3.2/29-7
0x00B0	MB3—Message buffer 3	R/W	0x0000_0000	29.3.2/29-7
0x00C0	MB4—Message buffer 4	R/W	0x0000_0000	29.3.2/29-7
0x00D0	MB5—Message buffer 5	R/W	0x0000_0000	29.3.2/29-7
0x00E0	MB6—Message buffer 6	R/W	0x0000_0000	29.3.2/29-7
0x00F0	MB7—Message buffer 7	R/W	0x0000_0000	29.3.2/29-7
0x0100	MB8—Message buffer 8	R/W	0x0000_0000	29.3.2/29-7
0x0110	MB9—Message buffer 9	R/W	0x0000_0000	29.3.2/29-7
0x0120	MB10—Message buffer 10	R/W	0x0000_0000	29.3.2/29-7
0x0130	MB11—Message buffer 11	R/W	0x0000_0000	29.3.2/29-7
0x0140	MB12—Message buffer 12	R/W	0x0000_0000	29.3.2/29-7
0x0150	MB13—Message buffer 13	R/W	0x0000_0000	29.3.2/29-7
0x0160	MB14—Message buffer 14	R/W	0x0000_0000	29.3.2/29-7
0x0170	MB15—Message buffer 15	R/W	0x0000_0000	29.3.2/29-7
0x0180	MB16—Message buffer 16	R/W	0x0000_0000	29.3.2/29-7
0x0190	MB17—Message buffer 17	R/W	0x0000_0000	29.3.2/29-7
0x01A0	MB18—Message buffer 18	R/W	0x0000_0000	29.3.2/29-7
0x01B0	MB19—Message buffer 19	R/W	0x0000_0000	29.3.2/29-7
0x01C0	MB20—Message buffer 20	R/W	0x0000_0000	29.3.2/29-7
0x01D0	MB21—Message buffer 21	R/W	0x0000_0000	29.3.2/29-7
0x01E0	MB22—Message buffer 22	R/W	0x0000_0000	29.3.2/29-7
0x01F0	MB23—Message buffer 23	R/W	0x0000_0000	29.3.2/29-7
0x0200	MB24—Message buffer 24	R/W	0x0000_0000	29.3.2/29-7
0x0210	MB25—Message buffer 25	R/W	0x0000_0000	29.3.2/29-7
0x0220	MB26—Message buffer 26	R/W	0x0000_0000	29.3.2/29-7
0x0230	MB27—Message buffer 27	R/W	0x0000_0000	29.3.2/29-7
0x0240	MB28—Message buffer 28	R/W	0x0000_0000	29.3.2/29-7
0x0250	MB29—Message buffer 29	R/W	0x0000_0000	29.3.2/29-7
0x0260	MB30—Message buffer 30	R/W	0x0000_0000	29.3.2/29-7
0x0270	MB31—Message buffer 31	R/W	0x0000_0000	29.3.2/29-7

Table A-4. PXN20 Detailed Register Map (continued)

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
0x0280	MB32—Message buffer 32	R/W	0x0000_0000	29.3.2/29-7
0x0290	MB33—Message buffer 33	R/W	0x0000_0000	29.3.2/29-7
0x02A0	MB34—Message buffer 34	R/W	0x0000_0000	29.3.2/29-7
0x02B0	MB35—Message buffer 35	R/W	0x0000_0000	29.3.2/29-7
0x02C0	MB36—Message buffer 36	R/W	0x0000_0000	29.3.2/29-7
0x02D0	MB37—Message buffer 37	R/W	0x0000_0000	29.3.2/29-7
0x02E0	MB38—Message buffer 38	R/W	0x0000_0000	29.3.2/29-7
0x02F0	MB39—Message buffer 39	R/W	0x0000_0000	29.3.2/29-7
0x0300	MB40—Message buffer 40	R/W	0x0000_0000	29.3.2/29-7
0x0310	MB41—Message buffer 41	R/W	0x0000_0000	29.3.2/29-7
0x0320	MB42—Message buffer 42	R/W	0x0000_0000	29.3.2/29-7
0x0330	MB43—Message buffer 43	R/W	0x0000_0000	29.3.2/29-7
0x0340	MB44—Message buffer 44	R/W	0x0000_0000	29.3.2/29-7
0x0350	MB45—Message buffer 45	R/W	0x0000_0000	29.3.2/29-7
0x0360	MB46—Message buffer 46	R/W	0x0000_0000	29.3.2/29-7
0x0370	MB47—Message buffer 47	R/W	0x0000_0000	29.3.2/29-7
0x0380	MB48—Message buffer 48	R/W	0x0000_0000	29.3.2/29-7
0x0390	MB49—Message buffer 49	R/W	0x0000_0000	29.3.2/29-7
0x03A0	MB50—Message buffer 50	R/W	0x0000_0000	29.3.2/29-7
0x03B0	MB51—Message buffer 51	R/W	0x0000_0000	29.3.2/29-7
0x03C0	MB52—Message buffer 52	R/W	0x0000_0000	29.3.2/29-7
0x03D0	MB53—Message buffer 53	R/W	0x0000_0000	29.3.2/29-7
0x03E0	MB54—Message buffer 54	R/W	0x0000_0000	29.3.2/29-7
0x03F0	MB55—Message buffer 55	R/W	0x0000_0000	29.3.2/29-7
0x0400	MB56—Message buffer 56	R/W	0x0000_0000	29.3.2/29-7
0x0410	MB57—Message buffer 57	R/W	0x0000_0000	29.3.2/29-7
0x0420	MB58—Message buffer 58	R/W	0x0000_0000	29.3.2/29-7
0x0430	MB59—Message buffer 59	R/W	0x0000_0000	29.3.2/29-7
0x0440	MB60—Message buffer 60	R/W	0x0000_0000	29.3.2/29-7
0x0450	MB61—Message buffer 61	R/W	0x0000_0000	29.3.2/29-7
0x0460	MB62—Message buffer 62	R/W	0x0000_0000	29.3.2/29-7
0x0470	MB63—Message buffer 63	R/W	0x0000_0000	29.3.2/29-7

Table A-4. PXN20 Detailed Register Map (continued)

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
0x0480–087F	Reserved			
0x0880	CANE_RXIMR0—Rx individual mask register 0	R/W	0x0000_0000	29.3.4.11/29-26
0x0884	CANE_RXIMR1—Rx individual mask register 1	R/W	0x0000_0000	29.3.4.11/29-26
0x0888	CANE_RXIMR2—Rx individual mask register 2	R/W	0x0000_0000	29.3.4.11/29-26
0x088C	CANE_RXIMR3—Rx individual mask register 3	R/W	0x0000_0000	29.3.4.11/29-26
0x0890	CANE_RXIMR4—Rx individual mask register 4	R/W	0x0000_0000	29.3.4.11/29-26
0x0894	CANE_RXIMR5—Rx individual mask register 5	R/W	0x0000_0000	29.3.4.11/29-26
0x0898	CANE_RXIMR6—Rx individual mask register 6	R/W	0x0000_0000	29.3.4.11/29-26
0x089C	CANE_RXIMR7—Rx individual mask register 7	R/W	0x0000_0000	29.3.4.11/29-26
0x08A0	CANE_RXIMR8—Rx individual mask register 8	R/W	0x0000_0000	29.3.4.11/29-26
0x08A4	CANE_RXIMR9—Rx individual mask register 9	R/W	0x0000_0000	29.3.4.11/29-26
0x08A8	CANE_RXIMR10—Rx individual mask register 10	R/W	0x0000_0000	29.3.4.11/29-26
0x08AC	CANE_RXIMR11—Rx individual mask register 11	R/W	0x0000_0000	29.3.4.11/29-26
0x08B0	CANE_RXIMR12—Rx individual mask register 12	R/W	0x0000_0000	29.3.4.11/29-26
0x08B4	CANE_RXIMR13—Rx individual mask register 13	R/W	0x0000_0000	29.3.4.11/29-26
0x08B8	CANE_RXIMR14—Rx individual mask register 14	R/W	0x0000_0000	29.3.4.11/29-26
0x08BC	CANE_RXIMR15—Rx individual mask register 15	R/W	0x0000_0000	29.3.4.11/29-26
0x08C0	CANE_RXIMR16—Rx individual mask register 16	R/W	0x0000_0000	29.3.4.11/29-26
0x08C4	CANE_RXIMR17—Rx individual mask register 17	R/W	0x0000_0000	29.3.4.11/29-26
0x08C8	CANE_RXIMR18—Rx individual mask register 18	R/W	0x0000_0000	29.3.4.11/29-26
0x08CC	CANE_RXIMR19—Rx individual mask register 19	R/W	0x0000_0000	29.3.4.11/29-26
0x08D0	CANE_RXIMR20—Rx individual mask register 20	R/W	0x0000_0000	29.3.4.11/29-26
0x08D4	CANE_RXIMR21—Rx individual mask register 21	R/W	0x0000_0000	29.3.4.11/29-26
0x08D8	CANE_RXIMR22—Rx individual mask register 22	R/W	0x0000_0000	29.3.4.11/29-26
0x08DC	CANE_RXIMR23—Rx individual mask register 23	R/W	0x0000_0000	29.3.4.11/29-26
0x08E0	CANE_RXIMR24—Rx individual mask register 24	R/W	0x0000_0000	29.3.4.11/29-26
0x08E4	CANE_RXIMR25—Rx individual mask register 25	R/W	0x0000_0000	29.3.4.11/29-26
0x08E8	CANE_RXIMR26—Rx individual mask register 26	R/W	0x0000_0000	29.3.4.11/29-26
0x08EC	CANE_RXIMR27—Rx individual mask register 27	R/W	0x0000_0000	29.3.4.11/29-26
0x08F0	CANE_RXIMR28—Rx individual mask register 28	R/W	0x0000_0000	29.3.4.11/29-26
0x08F4	CANE_RXIMR29—Rx individual mask register 29	R/W	0x0000_0000	29.3.4.11/29-26
0x08F8	CANE_RXIMR30—Rx individual mask register 30	R/W	0x0000_0000	29.3.4.11/29-26

Table A-4. PXN20 Detailed Register Map (continued)

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
0x08FC	CANE_RXIMR31—Rx individual mask register 31	R/W	0x0000_0000	29.3.4.11/29-26
0x0900	CANE_RXIMR32—Rx individual mask register 32	R/W	0x0000_0000	29.3.4.11/29-26
0x0904	CANE_RXIMR33—Rx individual mask register 33	R/W	0x0000_0000	29.3.4.11/29-26
0x0908	CANE_RXIMR34—Rx individual mask register 34	R/W	0x0000_0000	29.3.4.11/29-26
0x090C	CANE_RXIMR35—Rx individual mask register 35	R/W	0x0000_0000	29.3.4.11/29-26
0x0910	CANE_RXIMR36—Rx individual mask register 36	R/W	0x0000_0000	29.3.4.11/29-26
0x0914	CANE_RXIMR37—Rx individual mask register 37	R/W	0x0000_0000	29.3.4.11/29-26
0x0918	CANE_RXIMR38—Rx individual mask register 38	R/W	0x0000_0000	29.3.4.11/29-26
0x091C	CANE_RXIMR39—Rx individual mask register 39	R/W	0x0000_0000	29.3.4.11/29-26
0x0920	CANE_RXIMR40—Rx individual mask register 40	R/W	0x0000_0000	29.3.4.11/29-26
0x0924	CANE_RXIMR41—Rx individual mask register 41	R/W	0x0000_0000	29.3.4.11/29-26
0x0928	CANE_RXIMR42—Rx individual mask register 42	R/W	0x0000_0000	29.3.4.11/29-26
0x092C	CANE_RXIMR43—Rx individual mask register 43	R/W	0x0000_0000	29.3.4.11/29-26
0x0930	CANE_RXIMR44—Rx individual mask register 44	R/W	0x0000_0000	29.3.4.11/29-26
0x0934	CANE_RXIMR45—Rx individual mask register 45	R/W	0x0000_0000	29.3.4.11/29-26
0x0938	CANE_RXIMR46—Rx individual mask register 46	R/W	0x0000_0000	29.3.4.11/29-26
0x093C	CANE_RXIMR47—Rx individual mask register 47	R/W	0x0000_0000	29.3.4.11/29-26
0x0940	CANE_RXIMR48—Rx individual mask register 48	R/W	0x0000_0000	29.3.4.11/29-26
0x0944	CANE_RXIMR49—Rx individual mask register 49	R/W	0x0000_0000	29.3.4.11/29-26
0x0948	CANE_RXIMR50—Rx individual mask register 50	R/W	0x0000_0000	29.3.4.11/29-26
0x094C	CANE_RXIMR51—Rx individual mask register 51	R/W	0x0000_0000	29.3.4.11/29-26
0x0950	CANE_RXIMR52—Rx individual mask register 52	R/W	0x0000_0000	29.3.4.11/29-26
0x0954	CANE_RXIMR53—Rx individual mask register 53	R/W	0x0000_0000	29.3.4.11/29-26
0x0958	CANE_RXIMR54—Rx individual mask register 54	R/W	0x0000_0000	29.3.4.11/29-26
0x095C	CANE_RXIMR55—Rx individual mask register 55	R/W	0x0000_0000	29.3.4.11/29-26
0x0960	CANE_RXIMR56—Rx individual mask register 56	R/W	0x0000_0000	29.3.4.11/29-26
0x0964	CANE_RXIMR57—Rx individual mask register 57	R/W	0x0000_0000	29.3.4.11/29-26
0x0968	CANE_RXIMR58—Rx individual mask register 58	R/W	0x0000_0000	29.3.4.11/29-26
0x096C	CANE_RXIMR59—Rx individual mask register 59	R/W	0x0000_0000	29.3.4.11/29-26
0x0970	CANE_RXIMR60—Rx individual mask register 60	R/W	0x0000_0000	29.3.4.11/29-26
0x0974	CANE_RXIMR61—Rx individual mask register 61	R/W	0x0000_0000	29.3.4.11/29-26
0x0978	CANE_RXIMR62—Rx individual mask register 62	R/W	0x0000_0000	29.3.4.11/29-26

Table A-4. PXN20 Detailed Register Map (continued)

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
0x097C	CANE_RXIMR63—Rx individual mask register 63	R/W	0x0000_0000	29.3.4.11/29-26
0x0980–0x3FFF	Reserved			
0xFFFD_4000	**FlexCAN_F** **Chapter 29, Controller Area Network (FlexCAN)**			
0x0000	CANF_MCR—Module configuration register	R/W	0x9890_000F	29.3.4.1/29-11
0x0004	CANF_CTRL—control register	R/W	0x0000_0000	29.3.4.2/29-14
0x0008	CANF_TIMER—Free-running timer	R/W	0x0000_0000	29.3.4.3/29-17
0x000C	Reserved			
0x0010	CANF_RXGMASK—Rx global mask	R/W	0xFFFF_FFFF	29.3.4.4.1/29-18
0x0014	CANF_RX14MASK—Rx buffer 14 mask	R/W	0xFFFF_FFFF	29.3.4.4.2/29-19
0x0018	CANF_RX15MASK—Rx buffer 15 mask	R/W	0xFFFF_FFFF	29.3.4.4.3/29-19
0x001C	CANF_ECR—Error counter register	R/W	0x0000_0000	29.3.2/29-7
0x0020	CANF_ESR—Error and status register	R/W	0x0000_0000	29.3.4.6/29-21
0x0024	CANF_IMASK2—Interrupt masks 2	R/W	0x0000_0000	29.3.4.7/29-23
0x0028	CANF_IMASK1—Interrupt masks 1	R/W	0x0000_0000	29.3.4.8/29-24
0x002C	CANF_IFLAG2—Interrupt flags 2	R/W	0x0000_0000	29.3.4.9/29-24
0x0030	CANF_IFLAG1—Interrupt flags 1	R/W	0x0000_0000	29.3.4.10/29-25
0x0034–0x007F	Reserved			
0x0080	MB0—Message buffer 0	R/W	0x0000_0000	29.3.2/29-7
0x0090	MB1—Message buffer 1	R/W	0x0000_0000	29.3.2/29-7
0x00AO	MB2—Message buffer 2	R/W	0x0000_0000	29.3.2/29-7
0x00B0	MB3—Message buffer 3	R/W	0x0000_0000	29.3.2/29-7
0x00C0	MB4—Message buffer 4	R/W	0x0000_0000	29.3.2/29-7
0x00D0	MB5—Message buffer 5	R/W	0x0000_0000	29.3.2/29-7
0x00E0	MB6—Message buffer 6	R/W	0x0000_0000	29.3.2/29-7
0x00F0	MB7—Message buffer 7	R/W	0x0000_0000	29.3.2/29-7
0x0100	MB8—Message buffer 8	R/W	0x0000_0000	29.3.2/29-7
0x0110	MB9—Message buffer 9	R/W	0x0000_0000	29.3.2/29-7
0x0120	MB10—Message buffer 10	R/W	0x0000_0000	29.3.2/29-7
0x0130	MB11—Message buffer 11	R/W	0x0000_0000	29.3.2/29-7

Table A-4. PXN20 Detailed Register Map (continued)

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
0x0140	MB12—Message buffer 12	R/W	0x0000_0000	29.3.2/29-7
0x0150	MB13—Message buffer 13	R/W	0x0000_0000	29.3.2/29-7
0x0160	MB14—Message buffer 14	R/W	0x0000_0000	29.3.2/29-7
0x0170	MB15—Message buffer 15	R/W	0x0000_0000	29.3.2/29-7
0x0180	MB16—Message buffer 16	R/W	0x0000_0000	29.3.2/29-7
0x0190	MB17—Message buffer 17	R/W	0x0000_0000	29.3.2/29-7
0x01A0	MB18—Message buffer 18	R/W	0x0000_0000	29.3.2/29-7
0x01B0	MB19—Message buffer 19	R/W	0x0000_0000	29.3.2/29-7
0x01C0	MB20—Message buffer 20	R/W	0x0000_0000	29.3.2/29-7
0x01D0	MB21—Message buffer 21	R/W	0x0000_0000	29.3.2/29-7
0x01E0	MB22—Message buffer 22	R/W	0x0000_0000	29.3.2/29-7
0x01F0	MB23—Message buffer 23	R/W	0x0000_0000	29.3.2/29-7
0x0200	MB24—Message buffer 24	R/W	0x0000_0000	29.3.2/29-7
0x0210	MB25—Message buffer 25	R/W	0x0000_0000	29.3.2/29-7
0x0220	MB26—Message buffer 26	R/W	0x0000_0000	29.3.2/29-7
0x0230	MB27—Message buffer 27	R/W	0x0000_0000	29.3.2/29-7
0x0240	MB28—Message buffer 28	R/W	0x0000_0000	29.3.2/29-7
0x0250	MB29—Message buffer 29	R/W	0x0000_0000	29.3.2/29-7
0x0260	MB30—Message buffer 30	R/W	0x0000_0000	29.3.2/29-7
0x0270	MB31—Message buffer 31	R/W	0x0000_0000	29.3.2/29-7
0x0280	MB32—Message buffer 32	R/W	0x0000_0000	29.3.2/29-7
0x0290	MB33—Message buffer 33	R/W	0x0000_0000	29.3.2/29-7
0x02A0	MB34—Message buffer 34	R/W	0x0000_0000	29.3.2/29-7
0x02B0	MB35—Message buffer 35	R/W	0x0000_0000	29.3.2/29-7
0x02C0	MB36—Message buffer 36	R/W	0x0000_0000	29.3.2/29-7
0x02D0	MB37—Message buffer 37	R/W	0x0000_0000	29.3.2/29-7
0x02E0	MB38—Message buffer 38	R/W	0x0000_0000	29.3.2/29-7
0x02F0	MB39—Message buffer 39	R/W	0x0000_0000	29.3.2/29-7
0x0300	MB40—Message buffer 40	R/W	0x0000_0000	29.3.2/29-7
0x0310	MB41—Message buffer 41	R/W	0x0000_0000	29.3.2/29-7
0x0320	MB42—Message buffer 42	R/W	0x0000_0000	29.3.2/29-7
0x0330	MB43—Message buffer 43	R/W	0x0000_0000	29.3.2/29-7

Table A-4. PXN20 Detailed Register Map (continued)

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
0x0340	MB44—Message buffer 44	R/W	0x0000_0000	29.3.2/29-7
0x0350	MB45—Message buffer 45	R/W	0x0000_0000	29.3.2/29-7
0x0360	MB46—Message buffer 46	R/W	0x0000_0000	29.3.2/29-7
0x0370	MB47—Message buffer 47	R/W	0x0000_0000	29.3.2/29-7
0x0380	MB48—Message buffer 48	R/W	0x0000_0000	29.3.2/29-7
0x0390	MB49—Message buffer 49	R/W	0x0000_0000	29.3.2/29-7
0x03A0	MB50—Message buffer 50	R/W	0x0000_0000	29.3.2/29-7
0x03B0	MB51—Message buffer 51	R/W	0x0000_0000	29.3.2/29-7
0x03C0	MB52—Message buffer 52	R/W	0x0000_0000	29.3.2/29-7
0x03D0	MB53—Message buffer 53	R/W	0x0000_0000	29.3.2/29-7
0x03E0	MB54—Message buffer 54	R/W	0x0000_0000	29.3.2/29-7
0x03F0	MB55—Message buffer 55	R/W	0x0000_0000	29.3.2/29-7
0x0400	MB56—Message buffer 56	R/W	0x0000_0000	29.3.2/29-7
0x0410	MB57—Message buffer 57	R/W	0x0000_0000	29.3.2/29-7
0x0420	MB58—Message buffer 58	R/W	0x0000_0000	29.3.2/29-7
0x0430	MB59—Message buffer 59	R/W	0x0000_0000	29.3.2/29-7
0x0440	MB60—Message buffer 60	R/W	0x0000_0000	29.3.2/29-7
0x0450	MB61—Message buffer 61	R/W	0x0000_0000	29.3.2/29-7
0x0460	MB62—Message buffer 62	R/W	0x0000_0000	29.3.2/29-7
0x0470	MB63—Message buffer 63	R/W	0x0000_0000	29.3.2/29-7
0x0480–087F	Reserved			
0x0880	CANF_RXIMR0—Rx individual mask register 0	R/W	0x0000_0000	29.3.4.11/29-26
0x0884	CANF_RXIMR1—Rx individual mask register 1	R/W	0x0000_0000	29.3.4.11/29-26
0x0888	CANF_RXIMR2—Rx individual mask register 2	R/W	0x0000_0000	29.3.4.11/29-26
0x088C	CANF_RXIMR3—Rx individual mask register 3	R/W	0x0000_0000	29.3.4.11/29-26
0x0890	CANF_RXIMR4—Rx individual mask register 4	R/W	0x0000_0000	29.3.4.11/29-26
0x0894	CANF_RXIMR5—Rx individual mask register 5	R/W	0x0000_0000	29.3.4.11/29-26
0x0898	CANF_RXIMR6—Rx individual mask register 6	R/W	0x0000_0000	29.3.4.11/29-26
0x089C	CANF_RXIMR7—Rx individual mask register 7	R/W	0x0000_0000	29.3.4.11/29-26
0x08A0	CANF_RXIMR8—Rx individual mask register 8	R/W	0x0000_0000	29.3.4.11/29-26
0x08A4	CANF_RXIMR9—Rx individual mask register 9	R/W	0x0000_0000	29.3.4.11/29-26
0x08A8	CANF_RXIMR10—Rx individual mask register 10	R/W	0x0000_0000	29.3.4.11/29-26

Table A-4. PXN20 Detailed Register Map (continued)

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
0x08AC	CANF_RXIMR11—Rx individual mask register 11	R/W	0x0000_0000	29.3.4.11/29-26
0x08B0	CANF_RXIMR12—Rx individual mask register 12	R/W	0x0000_0000	29.3.4.11/29-26
0x08B4	CANF_RXIMR13—Rx individual mask register 13	R/W	0x0000_0000	29.3.4.11/29-26
0x08B8	CANF_RXIMR14—Rx individual mask register 14	R/W	0x0000_0000	29.3.4.11/29-26
0x08BC	CANF_RXIMR15—Rx individual mask register 15	R/W	0x0000_0000	29.3.4.11/29-26
0x08C0	CANF_RXIMR16—Rx individual mask register 16	R/W	0x0000_0000	29.3.4.11/29-26
0x08C4	CANF_RXIMR17—Rx individual mask register 17	R/W	0x0000_0000	29.3.4.11/29-26
0x08C8	CANF_RXIMR18—Rx individual mask register 18	R/W	0x0000_0000	29.3.4.11/29-26
0x08CC	CANF_RXIMR19—Rx individual mask register 19	R/W	0x0000_0000	29.3.4.11/29-26
0x08D0	CANF_RXIMR20—Rx individual mask register 20	R/W	0x0000_0000	29.3.4.11/29-26
0x08D4	CANF_RXIMR21—Rx individual mask register 21	R/W	0x0000_0000	29.3.4.11/29-26
0x08D8	CANF_RXIMR22—Rx individual mask register 22	R/W	0x0000_0000	29.3.4.11/29-26
0x08DC	CANF_RXIMR23—Rx individual mask register 23	R/W	0x0000_0000	29.3.4.11/29-26
0x08E0	CANF_RXIMR24—Rx individual mask register 24	R/W	0x0000_0000	29.3.4.11/29-26
0x08E4	CANF_RXIMR25—Rx individual mask register 25	R/W	0x0000_0000	29.3.4.11/29-26
0x08E8	CANF_RXIMR26—Rx individual mask register 26	R/W	0x0000_0000	29.3.4.11/29-26
0x08EC	CANF_RXIMR27—Rx individual mask register 27	R/W	0x0000_0000	29.3.4.11/29-26
0x08F0	CANF_RXIMR28—Rx individual mask register 28	R/W	0x0000_0000	29.3.4.11/29-26
0x08F4	CANF_RXIMR29—Rx individual mask register 29	R/W	0x0000_0000	29.3.4.11/29-26
0x08F8	CANF_RXIMR30—Rx individual mask register 30	R/W	0x0000_0000	29.3.4.11/29-26
0x08FC	CANF_RXIMR31—Rx individual mask register 31	R/W	0x0000_0000	29.3.4.11/29-26
0x0900	CANF_RXIMR32—Rx individual mask register 32	R/W	0x0000_0000	29.3.4.11/29-26
0x0904	CANF_RXIMR33—Rx individual mask register 33	R/W	0x0000_0000	29.3.4.11/29-26
0x0908	CANF_RXIMR34—Rx individual mask register 34	R/W	0x0000_0000	29.3.4.11/29-26
0x090C	CANF_RXIMR35—Rx individual mask register 35	R/W	0x0000_0000	29.3.4.11/29-26
0x0910	CANF_RXIMR36—Rx individual mask register 36	R/W	0x0000_0000	29.3.4.11/29-26
0x0914	CANF_RXIMR37—Rx individual mask register 37	R/W	0x0000_0000	29.3.4.11/29-26
0x0918	CANF_RXIMR38—Rx individual mask register 38	R/W	0x0000_0000	29.3.4.11/29-26
0x091C	CANF_RXIMR39—Rx individual mask register 39	R/W	0x0000_0000	29.3.4.11/29-26
0x0920	CANF_RXIMR40—Rx individual mask register 40	R/W	0x0000_0000	29.3.4.11/29-26
0x0924	CANF_RXIMR41—Rx individual mask register 41	R/W	0x0000_0000	29.3.4.11/29-26
0x0928	CANF_RXIMR42—Rx individual mask register 42	R/W	0x0000_0000	29.3.4.11/29-26

Table A-4. PXN20 Detailed Register Map (continued)

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
0x092C	CANF_RXIMR43—Rx individual mask register 43	R/W	0x0000_0000	29.3.4.11/29-26
0x0930	CANF_RXIMR44—Rx individual mask register 44	R/W	0x0000_0000	29.3.4.11/29-26
0x0934	CANF_RXIMR45—Rx individual mask register 45	R/W	0x0000_0000	29.3.4.11/29-26
0x0938	CANF_RXIMR46—Rx individual mask register 46	R/W	0x0000_0000	29.3.4.11/29-26
0x093C	CANF_RXIMR47—Rx individual mask register 47	R/W	0x0000_0000	29.3.4.11/29-26
0x0940	CANF_RXIMR48—Rx individual mask register 48	R/W	0x0000_0000	29.3.4.11/29-26
0x0944	CANF_RXIMR49—Rx individual mask register 49	R/W	0x0000_0000	29.3.4.11/29-26
0x0948	CANF_RXIMR50—Rx individual mask register 50	R/W	0x0000_0000	29.3.4.11/29-26
0x094C	CANF_RXIMR51—Rx individual mask register 51	R/W	0x0000_0000	29.3.4.11/29-26
0x0950	CANF_RXIMR52—Rx individual mask register 52	R/W	0x0000_0000	29.3.4.11/29-26
0x0954	CANF_RXIMR53—Rx individual mask register 53	R/W	0x0000_0000	29.3.4.11/29-26
0x0958	CANF_RXIMR54—Rx individual mask register 54	R/W	0x0000_0000	29.3.4.11/29-26
0x095C	CANF_RXIMR55—Rx individual mask register 55	R/W	0x0000_0000	29.3.4.11/29-26
0x0960	CANF_RXIMR56—Rx individual mask register 56	R/W	0x0000_0000	29.3.4.11/29-26
0x0964	CANF_RXIMR57—Rx individual mask register 57	R/W	0x0000_0000	29.3.4.11/29-26
0x0968	CANF_RXIMR58—Rx individual mask register 58	R/W	0x0000_0000	29.3.4.11/29-26
0x096C	CANF_RXIMR59—Rx individual mask register 59	R/W	0x0000_0000	29.3.4.11/29-26
0x0970	CANF_RXIMR60—Rx individual mask register 60	R/W	0x0000_0000	29.3.4.11/29-26
0x0974	CANF_RXIMR61—Rx individual mask register 61	R/W	0x0000_0000	29.3.4.11/29-26
0x0978	CANF_RXIMR62—Rx individual mask register 62	R/W	0x0000_0000	29.3.4.11/29-26
0x097C	CANF_RXIMR63—Rx individual mask register 63	R/W	0x0000_0000	29.3.4.11/29-26
0x0980–0x3FFF	Reserved			
0xFFFD_8000	**CTU** **Chapter 33, Cross Triggering Unit (CTU)**			
0x0000	CTU_CSR – Control status register	R/W	0x0000_0000	33.4.1.1/33-4
0x0004	CTU_SVR1 – Start value register 1	R/W	0x0000_0000	33.4.1.2/33-5
0x0008	CTU_SVR2 – Start value register 2	R/W	0x0000_0000	33.4.1.2/33-5
0x000C	CTU_SVR3 – Start value register 3	R/W	0x0000_0000	33.4.1.2/33-5
0x0010	CTU_SVR4 – Start value register 4	R/W	0x0000_0000	33.4.1.2/33-5
0x0014	CTU_SVR5 – Start value register 5	R/W	0x0000_0000	33.4.1.2/33-5
0x0018	CTU_SVR6 – Start value register 6	R/W	0x0000_0000	33.4.1.2/33-5
0x001C	CTU_SVR7 – Start value register 7	R/W	0x0000_0000	33.4.1.2/33-5

Table A-4. PXN20 Detailed Register Map (continued)

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
0x0020	CTU_CVR0 – current value register 0	R/W	0x0000_0000	33.4.1.3/33-5
0x0024	CTU_CVR1 – current value register 1	R/W	0x0000_0000	33.4.1.3/33-5
0x0028	CTU_CVR2 – current value register 2	R/W	0x0000_0000	33.4.1.3/33-5
0x002C	CTU_CVR3 – current value register 3	R/W	0x0000_0000	33.4.1.3/33-5
0x0030	CTU_EVTCFGR0 – Event configuration register 0	R/W	0x0000_0000	33.4.1.4/33-6
0x0034	CTU_EVTCFGR1 – Event configuration register 1	R/W	0x0000_0000	33.4.1.4/33-6
0x0038	CTU_EVTCFGR2 – Event configuration register 2	R/W	0x0000_0000	33.4.1.4/33-6
0x003C	CTU_EVTCFGR3 – Event configuration register 3	R/W	0x0000_0000	33.4.1.4/33-6
0x0040	CTU_EVTCFGR4 – Event configuration register 4	R/W	0x0000_0000	33.4.1.4/33-6
0x0044	CTU_EVTCFGR5 – Event configuration register 5	R/W	0x0000_0000	33.4.1.4/33-6
0x0048	CTU_EVTCFGR6 – Event configuration register 6	R/W	0x0000_0000	33.4.1.4/33-6
0x004C	CTU_EVTCFGR7 – Event configuration register 7	R/W	0x0000_0000	33.4.1.4/33-6
0x0050	CTU_EVTCFGR8 – Event configuration register 8	R/W	0x0000_0000	33.4.1.4/33-6
0x0054	CTU_EVTCFGR9 – Event configuration register 9	R/W	0x0000_0000	33.4.1.4/33-6
0x0058	CTU_EVTCFGR10 – Event configuration register 10	R/W	0x0000_0000	33.4.1.4/33-6
0x005C	CTU_EVTCFGR11 – Event configuration register 11	R/W	0x0000_0000	33.4.1.4/33-6
0x0060	CTU_EVTCFGR12 – Event configuration register 12	R/W	0x0000_0000	33.4.1.4/33-6
0x0064	CTU_EVTCFGR13 – Event configuration register 13	R/W	0x0000_0000	33.4.1.4/33-6
0x0068	CTU_EVTCFGR14 – Event configuration register 14	R/W	0x0000_0000	33.4.1.4/33-6
0x006C	CTU_EVTCFGR15 – Event configuration register 15	R/W	0x0000_0000	33.4.1.4/33-6
0x0070	CTU_EVTCFGR16 – Event configuration register 16	R/W	0x0000_0000	33.4.1.4/33-6
0x0074	CTU_EVTCFGR17 – Event configuration register 17	R/W	0x0000_0000	33.4.1.4/33-6
0x0078	CTU_EVTCFGR18 – Event configuration register 18	R/W	0x0000_0000	33.4.1.4/33-6
0x007C	CTU_EVTCFGR19 – Event configuration register 19	R/W	0x0000_0000	33.4.1.4/33-6
0x0080	CTU_EVTCFGR20 – Event configuration register 20	R/W	0x0000_0000	33.4.1.4/33-6
0x0084	CTU_EVTCFGR21 – Event configuration register 21	R/W	0x0000_0000	33.4.1.4/33-6
0x0088	CTU_EVTCFGR22 – Event configuration register 22	R/W	0x0000_0000	33.4.1.4/33-6
0x008C	CTU_EVTCFGR23 – Event configuration register 23	R/W	0x0000_0000	33.4.1.4/33-6
0x0090	CTU_EVTCFGR24 – Event configuration register 24	R/W	0x0000_0000	33.4.1.4/33-6
0x0094	CTU_EVTCFGR25 – Event configuration register 25	R/W	0x0000_0000	33.4.1.4/33-6
0x0098	CTU_EVTCFGR26 – Event configuration register 26	R/W	0x0000_0000	33.4.1.4/33-6
0x009C	CTU_EVTCFGR27 – Event configuration register 27	R/W	0x0000_0000	33.4.1.4/33-6

Table A-4. PXN20 Detailed Register Map (continued)

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
0x00A0	CTU_EVTCFGR28 – Event configuration register 28	R/W	0x0000_0000	33.4.1.4/33-6
0x00A4	CTU_EVTCFGR29 – Event configuration register 29	R/W	0x0000_0000	33.4.1.4/33-6
0x00A8	CTU_EVTSELR30 – Event configuration register 30	R/W	0x0000_0000	33.4.1.4/33-6
0x00AC	CTU_EVTSELR31 – Event configuration register 31	R/W	0x0000_0000	33.4.1.4/33-6
0x00B0	CTU_EVTSELR32 – Event configuration register 32	R/W	0x0000_0000	33.4.1.4/33-6
0x00B4–0x3FFF	Reserved			
0xFFFD_C000	**DMA_MUX** **Chapter 23, DMA Channel Multiplexer (DMA_MUX)**			
0x0000	CHCONFIG0—Channel #0 configuration	R/W	0x00	23.3.2.1/23-4
0x0001	CHCONFIG1—Channel #1 configuration	R/W	0x00	23.3.2.1/23-4
0x0002	CHCONFIG2—Channel #2 configuration	R/W	0x00	23.3.2.1/23-4
0x0003	CHCONFIG3—Channel #3 configuration	R/W	0x00	23.3.2.1/23-4
0x0004	CHCONFIG4—Channel #4 configuration	R/W	0x00	23.3.2.1/23-4
0x0005	CHCONFIG5—Channel #5 configuration	R/W	0x00	23.3.2.1/23-4
0x0006	CHCONFIG6—Channel #6 configuration	R/W	0x00	23.3.2.1/23-4
0x0007	CHCONFIG7—Channel #7 configuration	R/W	0x00	23.3.2.1/23-4
0x0008	CHCONFIG8—Channel #8 configuration	R/W	0x00	23.3.2.1/23-4
0x0009	CHCONFIG9—Channel #9 configuration	R/W	0x00	23.3.2.1/23-4
0x000A	CHCONFIG10—Channel #10 configuration	R/W	0x00	23.3.2.1/23-4
0x000B	CHCONFIG11—Channel #11 configuration	R/W	0x00	23.3.2.1/23-4
0x000C	CHCONFIG12—Channel #12 configuration	R/W	0x00	23.3.2.1/23-4
0x000D	CHCONFIG13—Channel #13 configuration	R/W	0x00	23.3.2.1/23-4
0x000E	CHCONFIG14—Channel #14 configuration	R/W	0x00	23.3.2.1/23-4
0x000F	CHCONFIG15—Channel #15 configuration	R/W	0x00	23.3.2.1/23-4
0x0010	CHCONFIG16—Channel #16 configuration	R/W	0x00	23.3.2.1/23-4
0x0011	CHCONFIG17—Channel #17 configuration	R/W	0x00	23.3.2.1/23-4
0x0012	CHCONFIG18—Channel #18 configuration	R/W	0x00	23.3.2.1/23-4
0x0013	CHCONFIG19—Channel #19 configuration	R/W	0x00	23.3.2.1/23-4
0x0014	CHCONFIG20—Channel #20 configuration	R/W	0x00	23.3.2.1/23-4
0x0015	CHCONFIG21—Channel #21 configuration	R/W	0x00	23.3.2.1/23-4
0x0016	CHCONFIG22—Channel #22 configuration	R/W	0x00	23.3.2.1/23-4
0x0017	CHCONFIG23—Channel #23 configuration	R/W	0x00	23.3.2.1/23-4

Table A-4. PXN20 Detailed Register Map (continued)

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
0x0018	CHCONFIG24—Channel #24 configuration	R/W	0x00	23.3.2.1/23-4
0x0019	CHCONFIG25—Channel #25 configuration	R/W	0x00	23.3.2.1/23-4
0x001A	CHCONFIG26—Channel #26 configuration	R/W	0x00	23.3.2.1/23-4
0x001B	CHCONFIG27—Channel #27 configuration	R/W	0x00	23.3.2.1/23-4
0x001C	CHCONFIG28—Channel #28 configuration	R/W	0x00	23.3.2.1/23-4
0x001D	CHCONFIG29—Channel #29 configuration	R/W	0x00	23.3.2.1/23-4
0x001E	CHCONFIG30—Channel #30 configuration	R/W	0x00	23.3.2.1/23-4
0x001F	CHCONFIG31—Channel #31 configuration	R/W	0x00	23.3.2.1/23-4
0x0020–0x3FFF	Reserved			
0xFFFE_0000	**PIT** **Chapter 22, Periodic Interrupt Timer (PIT)**			
0x0000	PITMCR—PIT module control register	R/W	0x0000_0002	22.3.2.1/22-4
0x0004–0x00FF	Reserved			
0x0100	LDVAL1—Timer 1 load value register	R/W	0x0000_0000	22.3.2.2/22-5
0x0104	CVAL1—Timer 1 current value register	R/W	0x0000_0000	22.3.2.3/22-5
0x0108	TCTRL1—Timer 1 control register	R/W	0x0000_0000	22.3.2.4/22-6
0x010C	TFLG1—Timer 1 flag register	R/W	0x0000_0000	22.3.2.5/22-7
0x0110	LDVAL2—Timer 2 load value register	R/W	0x0000_0000	22.3.2.2/22-5
0x0114	CVAL2—Timer 2 current value register	R/W	0x0000_0000	22.3.2.3/22-5
0x0118	TCTRL2—Timer 2 control register	R/W	0x0000_0000	22.3.2.4/22-6
0x011C	TFLG2—Timer 2 flag register	R/W	0x0000_0000	22.3.2.5/22-7
0x0120	LDVAL3—Timer 3 load value register	R/W	0x0000_0000	22.3.2.2/22-5
0x0124	CVAL3—Timer 3 current value register	R/W	0x0000_0000	22.3.2.3/22-5
0x0128	TCTRL3—Timer 3 control register	R/W	0x0000_0000	22.3.2.4/22-6
0x012C	TFLG3—Timer 3 flag register	R/W	0x0000_0000	22.3.2.5/22-7
0x0130	LDVAL4—Timer 4 load value register	R/W	0x0000_0000	22.3.2.2/22-5
0x0134	CVAL4—Timer 4 current value register	R/W	0x0000_0000	22.3.2.3/22-5
0x0138	TCTRL4—Timer 4 control register	R/W	0x0000_0000	22.3.2.4/22-6
0x013C	TFLG4—Timer 4 flag register	R/W	0x0000_0000	22.3.2.5/22-7
0x0140	LDVAL5—Timer 5 load value register	R/W	0x0000_0000	22.3.2.2/22-5
0x0144	CVAL5—Timer 5 current value register	R/W	0x0000_0000	22.3.2.3/22-5
0x0148	TCTRL5—Timer 5 control register	R/W	0x0000_0000	22.3.2.4/22-6

Table A-4. PXN20 Detailed Register Map (continued)

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
0x014C	TFLG5—Timer 5 flag register	R/W	0x0000_0000	22.3.2.5/22-7
0x0150	LDVAL6—Timer 6 load value register	R/W	0x0000_0000	22.3.2.2/22-5
0x0154	CVAL6—Timer 6 current value register	R/W	0x0000_0000	22.3.2.3/22-5
0x0158	TCTRL6—Timer 6 control register	R/W	0x0000_0000	22.3.2.4/22-6
0x015C	TFLG6—Timer 6 flag register	R/W	0x0000_0000	22.3.2.5/22-7
0x0160	LDVAL7—Timer 7 load value register	R/W	0x0000_0000	22.3.2.2/22-5
0x0164	CVAL7—Timer 7 current value register	R/W	0x0000_0000	22.3.2.3/22-5
0x0168	TCTRL7—Timer 7 control register	R/W	0x0000_0000	22.3.2.4/22-6
0x016C	TFLG7—Timer 7 flag register	R/W	0x0000_0000	22.3.2.5/22-7
0x0170	LDVAL8—Timer 8 load value register	R/W	0x0000_0000	22.3.2.2/22-5
0x0174	CVAL8—Timer 8 current value register	R/W	0x0000_0000	22.3.2.3/22-5
0x0178	TCTRL8—Timer 8 control register	R/W	0x0000_0000	22.3.2.4/22-6
0x017C	TFLG8—Timer 8 flag register	R/W	0x0000_0000	22.3.2.5/22-7
0x0180–0x3FFF	Reserved			
0xFFFE_4000	**eMIOS_A** **Chapter 28, Enhanced Modular Input/Output Subsystem (eMIOS200)**			
0x0000	EMIOS_MCR—Module configuration register	R/W	0x0000_0000	28.3.2.1/28-9
0x0004	EMIOS_GFR—Global flag register	R	0x0000_0000	28.3.2.2/28-10
0x0008	EMIOS_OUDR—Output update disable register	R/W	0x0000_0000	28.3.2.3/28-11
0x000C	EMIOS_UCDIS—Stop (disable) channel register	R/W	0x0000_0000	28.3.2.4/28-11
0x0010–0x001F	Reserved			
0x0020	EMIOS_CADR[0]—Channel A data register	R/W	0x0000_0000	28.3.2.5/28-12
0x0024	EMIOS_CBDR[0]—Channel B data register	R/W	0x0000_0000	28.3.2.6/28-12
0x0028	EMIOS_CCNTR[0]—Channel counter register	R	0x0000_0000	28.3.2.7/28-13
0x002C	EMIOS_CCR[0]—Channel control register	R/W	0x0000_0000	28.3.2.8/28-14
0x0030	EMIOS_CSR[0]—Channel status register	R	0x0000_0000	28.3.2.9/28-19
0x0034	EMIOS_ALTA[0]—Alternate A register	R/W	0x0000_0000	28.3.2.10/28-19
0x0038–0x003F	Reserved			
0x0040	EMIOS_CADR[1]—Channel A data register	R/W	0x0000_0000	28.3.2.5/28-12
0x0044	EMIOS_CBDR[1]—Channel B data register	R/W	0x0000_0000	28.3.2.6/28-12
0x0048	EMIOS_CCNTR[1]—Counter register	R	0x0000_0000	28.3.2.7/28-13
0x004C	EMIOS_CCR[1]—Control register	R/W	0x0000_0000	28.3.2.8/28-14

Table A-4. PXN20 Detailed Register Map (continued)

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
0x0050	EMIOS_CSR[1]—Status register	R	0x0000_0000	28.3.2.9/28-19
0x0054	EMIOS_ALTA[1]—Alternate A register	R/W	0x0000_0000	28.3.2.10/28-19
0x0058–0x005F	Reserved			
0x0060	EMIOS_CADR[2]—Channel A data register	R/W	0x0000_0000	28.3.2.5/28-12
0x0064	EMIOS_CBDR[2]—Channel B data register	R/W	0x0000_0000	28.3.2.6/28-12
0x0068	EMIOS_CCNTR[2]—Counter register	R	0x0000_0000	28.3.2.7/28-13
0x006C	EMIOS_CCR[2]—Control register	R/W	0x0000_0000	28.3.2.8/28-14
0x0070	EMIOS_CSR[2]—Status register	R	0x0000_0000	28.3.2.9/28-19
0x0074	EMIOS_ALTA[2]—Alternate A register	R/W	0x0000_0000	28.3.2.10/28-19
0x0078–0x007F	Reserved			
0x0080	EMIOS_CADR[3]—Channel A data register	R/W	0x0000_0000	28.3.2.5/28-12
0x0084	EMIOS_CBDR[3]—Channel B data register	R/W	0x0000_0000	28.3.2.6/28-12
0x0086	EMIOS_CCNTR[3]—Counter register	R	0x0000_0000	28.3.2.7/28-13
0x008C	EMIOS_CCR[3]—Control register	R/W	0x0000_0000	28.3.2.8/28-14
0x0090	EMIOS_CSR[3]—Status register	R	0x0000_0000	28.3.2.9/28-19
0x0094	EMIOS_ALTA[3]—Alternate A register	R/W	0x0000_0000	28.3.2.10/28-19
0x0098–0x009F	Reserved			
0x00A0	EMIOS_CADR[4]—Channel A data register	R/W	0x0000_0000	28.3.2.5/28-12
0x00A4	EMIOS_CBDR[4]—Channel B data register	R/W	0x0000_0000	28.3.2.6/28-12
0x00A6	EMIOS_CCNTR[4]—Counter register	R	0x0000_0000	28.3.2.7/28-13
0x00AC	EMIOS_CCR[4]—Control register	R/W	0x0000_0000	28.3.2.8/28-14
0x00B0	EMIOS_CSR[4]—Status register	R	0x0000_0000	28.3.2.9/28-19
0x00B4	EMIOS_ALTA[4]—Alternate A register	R/W	0x0000_0000	28.3.2.10/28-19
0x00B8–0x00BF	Reserved			
0x00C0	EMIOS_CADR[5]—Channel A data register	R/W	0x0000_0000	28.3.2.5/28-12
0x00C4	EMIOS_CBDR[5]—Channel B data register	R/W	0x0000_0000	28.3.2.6/28-12
0x00C6	EMIOS_CCNTR[5]—Counter register	R	0x0000_0000	28.3.2.7/28-13
0x00CC	EMIOS_CCR[5]—Control register	R/W	0x0000_0000	28.3.2.8/28-14
0x00D0	EMIOS_CSR[5]—Status register	R	0x0000_0000	28.3.2.9/28-19
0x00D4	EMIOS_ALTA[5]—Alternate A register	R/W	0x0000_0000	28.3.2.10/28-19
0x00D8–0x00DF	Reserved			
0x00E0	EMIOS_CADR[6]—Channel A data register	R/W	0x0000_0000	28.3.2.5/28-12

Table A-4. PXN20 Detailed Register Map (continued)

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
0x00E4	EMIOS_CBDR[6]—Channel B data register	R/W	0x0000_0000	28.3.2.6/28-12
0x00E6	EMIOS_CCNTR[6]—Counter register	R	0x0000_0000	28.3.2.7/28-13
0x00EC	EMIOS_CCR[6]—Control register	R/W	0x0000_0000	28.3.2.8/28-14
0x00F0	EMIOS_CSR[6]—Status register	R	0x0000_0000	28.3.2.9/28-19
0x00F4	EMIOS_ALTA[6]—Alternate A register	R/W	0x0000_0000	28.3.2.10/28-19
0x00F8–0x00FF	Reserved			
0x0100	EMIOS_CADR[7]—Channel A data register	R/W	0x0000_0000	28.3.2.5/28-12
0x0104	EMIOS_CBDR[7]—Channel B data register	R/W	0x0000_0000	28.3.2.6/28-12
0x0106	EMIOS_CCNTR[7]—Counter register	R	0x0000_0000	28.3.2.7/28-13
0x010C	EMIOS_CCR[7]—Control register	R/W	0x0000_0000	28.3.2.8/28-14
0x0110	EMIOS_CSR[7]—Status register	R	0x0000_0000	28.3.2.9/28-19
0x0114	EMIOS_ALTA[7]—Alternate A register	R/W	0x0000_0000	28.3.2.10/28-19
0x0118–0x011F	Reserved			
0x0120	EMIOS_CADR[8]—Channel A data register	R/W	0x0000_0000	28.3.2.5/28-12
0x0124	EMIOS_CBDR[8]—Channel B data register	R/W	0x0000_0000	28.3.2.6/28-12
0x0126	EMIOS_CCNTR[8]—Counter register	R	0x0000_0000	28.3.2.7/28-13
0x012C	EMIOS_CCR[8]—Control register	R/W	0x0000_0000	28.3.2.8/28-14
0x0130	EMIOS_CSR[8]—Status register	R	0x0000_0000	28.3.2.9/28-19
0x0134	EMIOS_ALTA[8]—Alternate A register	R/W	0x0000_0000	28.3.2.10/28-19
0x0138–0x013F	Reserved			
0x0140	EMIOS_CADR[9]—Channel A data register	R/W	0x0000_0000	28.3.2.5/28-12
0x0144	EMIOS_CBDR[9]—Channel B data register	R/W	0x0000_0000	28.3.2.6/28-12
0x0146	EMIOS_CCNTR[9]—Counter register	R	0x0000_0000	28.3.2.7/28-13
0x014C	EMIOS_CCR[9]—Control register	R/W	0x0000_0000	28.3.2.8/28-14
0x0150	EMIOS_CSR[9]—Status register	R	0x0000_0000	28.3.2.9/28-19
0x0154	EMIOS_ALTA[9]—Alternate A register	R/W	0x0000_0000	28.3.2.10/28-19
0x0158–0x015F	Reserved			
0x0160	EMIOS_CADR[10]—Channel A data register	R/W	0x0000_0000	28.3.2.5/28-12
0x0164	EMIOS_CBDR[10]—Channel B data register	R/W	0x0000_0000	28.3.2.6/28-12
0x0166	EMIOS_CCNTR[10]—Counter register	R	0x0000_0000	28.3.2.7/28-13
0x016C	EMIOS_CCR[10]—Control register	R/W	0x0000_0000	28.3.2.8/28-14
0x0170	EMIOS_CSR[10]—Status register	R	0x0000_0000	28.3.2.9/28-19

Table A-4. PXN20 Detailed Register Map (continued)

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
0x0174	EMIOS_ALTA[10]—Alternate A register	R/W	0x0000_0000	28.3.2.10/28-19
0x0178–0x017F	Reserved			
0x0180	EMIOS_CADR[11]—Channel A data register	R/W	0x0000_0000	28.3.2.5/28-12
0x0184	EMIOS_CBDR[11]—Channel B data register	R/W	0x0000_0000	28.3.2.6/28-12
0x0168	EMIOS_CCNTR[11]—Counter register	R	0x0000_0000	28.3.2.7/28-13
0x018C	EMIOS_CCR[11]—Control register	R/W	0x0000_0000	28.3.2.8/28-14
0x0190	EMIOS_CSR[11]—Status register	R	0x0000_0000	28.3.2.9/28-19
0x0194	EMIOS_ALTA[11]—Alternate A register	R/W	0x0000_0000	28.3.2.10/28-19
0x0198–0x019F	Reserved			
0x01A0	EMIOS_CADR[12]—Channel A data register	R/W	0x0000_0000	28.3.2.5/28-12
0x01A4	EMIOS_CBDR[12]—Channel B data register	R/W	0x0000_0000	28.3.2.6/28-12
0x01A6	EMIOS_CCNTR[12]—Counter register	R	0x0000_0000	28.3.2.7/28-13
0x01AC	EMIOS_CCR[12]—Control register	R/W	0x0000_0000	28.3.2.8/28-14
0x01B0	EMIOS_CSR[12]—Status register	R	0x0000_0000	28.3.2.9/28-19
0x01B4	EMIOS_ALTA[12]—Alternate A register	R/W	0x0000_0000	28.3.2.10/28-19
0x01B8–0x01BF	Reserved			
0x01C0	EMIOS_CADR[13]—Channel A data register	R/W	0x0000_0000	28.3.2.5/28-12
0x01C4	EMIOS_CBDR[13]—Channel B data register	R/W	0x0000_0000	28.3.2.6/28-12
0x01C6	EMIOS_CCNTR[13]—Counter register	R	0x0000_0000	28.3.2.7/28-13
0x01CC	EMIOS_CCR[13]—Control register	R/W	0x0000_0000	28.3.2.8/28-14
0x01D0	EMIOS_CSR[13]—Status register	R	0x0000_0000	28.3.2.9/28-19
0x01D4	EMIOS_ALTA[13]—Alternate A register	R/W	0x0000_0000	28.3.2.10/28-19
0x01D8–0x01DF	Reserved			
0x01E0	EMIOS_CADR[14]—Channel A data register	R/W	0x0000_0000	28.3.2.5/28-12
0x01E4	EMIOS_CBDR[14]—Channel B data register	R/W	0x0000_0000	28.3.2.6/28-12
0x01E6	EMIOS_CCNTR[14]—Counter register	R	0x0000_0000	28.3.2.7/28-13
0x01EC	EMIOS_CCR[14]—Control register	R/W	0x0000_0000	28.3.2.8/28-14
0x01F0	EMIOS_CSR[14]—Status register	R	0x0000_0000	28.3.2.9/28-19
0x01F4	EMIOS_ALTA[14]—Alternate A register	R/W	0x0000_0000	28.3.2.10/28-19
0x01F8–0x01FF	Reserved			
0x0200	EMIOS_CADR[15]—Channel A data register	R/W	0x0000_0000	28.3.2.5/28-12
0x0204	EMIOS_CBDR[15]—Channel B data register	R/W	0x0000_0000	28.3.2.6/28-12

Table A-4. PXN20 Detailed Register Map (continued)

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
0x0206	EMIOS_CCNTR[15]—Counter register	R	0x0000_0000	28.3.2.7/28-13
0x020C	EMIOS_CCR[15]—Control register	R/W	0x0000_0000	28.3.2.8/28-14
0x0210	EMIOS_CSR[15]—Status register	R	0x0000_0000	28.3.2.9/28-19
0x0214	EMIOS_ALTA[15]—Alternate A register	R/W	0x0000_0000	28.3.2.10/28-19
0x0218–0x021F	Reserved			
0x0220	EMIOS_CADR[16]—Channel A data register	R/W	0x0000_0000	28.3.2.5/28-12
0x0224	EMIOS_CBDR[16]—Channel B data register	R/W	0x0000_0000	28.3.2.6/28-12
0x0226	EMIOS_CCNTR[16]—Counter register	R	0x0000_0000	28.3.2.7/28-13
0x022C	EMIOS_CCR[16]—Control register	R/W	0x0000_0000	28.3.2.8/28-14
0x0230	EMIOS_CSR[16]—Status register	R	0x0000_0000	28.3.2.9/28-19
0x0234	EMIOS_ALTA[16]—Alternate A register	R/W	0x0000_0000	28.3.2.10/28-19
0x0238–0x023F	Reserved			
0x0240	EMIOS_CADR[17]—Channel A data register	R/W	0x0000_0000	28.3.2.5/28-12
0x0244	EMIOS_CBDR[17]—Channel B data register	R/W	0x0000_0000	28.3.2.6/28-12
0x0248	EMIOS_CCNTR[17]—Counter register	R	0x0000_0000	28.3.2.7/28-13
0x024C	EMIOS_CCR[17]—Control register	R/W	0x0000_0000	28.3.2.8/28-14
0x0250	EMIOS_CSR[17—Status register	R	0x0000_0000	28.3.2.9/28-19
0x0254	EMIOS_ALTA[17]—Alternate A register	R/W	0x0000_0000	28.3.2.10/28-19
0x0258–0x025F	Reserved			
0x0260	EMIOS_CADR[18]—Channel A data register	R/W	0x0000_0000	28.3.2.5/28-12
0x0264	EMIOS_CBDR[18]—Channel B data register	R/W	0x0000_0000	28.3.2.6/28-12
0x0268	EMIOS_CCNTR[18]—Counter register	R	0x0000_0000	28.3.2.7/28-13
0x026C	EMIOS_CCR[18]—Control register	R/W	0x0000_0000	28.3.2.8/28-14
0x0270	EMIOS_CSR[18]—Status register	R	0x0000_0000	28.3.2.9/28-19
0x0274	EMIOS_ALTA[18]—Alternate A register	R/W	0x0000_0000	28.3.2.10/28-19
0x0278–0x027F	Reserved			
0x0280	EMIOS_CADR[19]—Channel A data register	R/W	0x0000_0000	28.3.2.5/28-12
0x0284	EMIOS_CBDR[19]—Channel B data register	R/W	0x0000_0000	28.3.2.6/28-12
0x0288	EMIOS_CCNTR[19]—Counter register	R	0x0000_0000	28.3.2.7/28-13
0x028C	EMIOS_CCR[19]—Control register	R/W	0x0000_0000	28.3.2.8/28-14
0x0290	EMIOS_CSR[19]—Status register	R	0x0000_0000	28.3.2.9/28-19
0x0294	EMIOS_ALTA[19]—Alternate A register	R/W	0x0000_0000	28.3.2.10/28-19

Table A-4. PXN20 Detailed Register Map (continued)

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
0x0298–0x029F	Reserved			
0x02A0	EMIOS_CADR[20]—Channel A data register	R/W	0x0000_0000	28.3.2.5/28-12
0x02A4	EMIOS_CBDR[20]—Channel B data register	R/W	0x0000_0000	28.3.2.6/28-12
0x02A8	EMIOS_CCNTR[20]—Counter register	R	0x0000_0000	28.3.2.7/28-13
0x02AC	EMIOS_CCR[20]—Control register	R/W	0x0000_0000	28.3.2.8/28-14
0x02B0	EMIOS_CSR[20]—Status register	R	0x0000_0000	28.3.2.9/28-19
0x02B4	EMIOS_ALTA[20]—Alternate A register	R/W	0x0000_0000	28.3.2.10/28-19
0x02B8–0x02BF	Reserved			
0x02C0	EMIOS_CADR[21]—Channel A data register	R/W	0x0000_0000	28.3.2.5/28-12
0x02C4	EMIOS_CBDR[21]—Channel B data register	R/W	0x0000_0000	28.3.2.6/28-12
0x02C8	EMIOS_CCNTR[21]—Counter register	R	0x0000_0000	28.3.2.7/28-13
0x02CC	EMIOS_CCR[21]—Control register	R/W	0x0000_0000	28.3.2.8/28-14
0x02D0	EMIOS_CSR[21]—Status register	R	0x0000_0000	28.3.2.9/28-19
0x02D4	EMIOS_ALTA[21]—Alternate A register	R/W	0x0000_0000	28.3.2.10/28-19
0x02D8–0x02DF	Reserved			
0x02E0	EMIOS_CADR[22]—Channel A data register	R/W	0x0000_0000	28.3.2.5/28-12
0x02E4	EMIOS_CBDR[22]—Channel B data register	R/W	0x0000_0000	28.3.2.6/28-12
0x02E8	EMIOS_CCNTR[22]—Counter register	R	0x0000_0000	28.3.2.7/28-13
0x02EC	EMIOS_CCR[22]—Control register	R/W	0x0000_0000	28.3.2.8/28-14
0x02F0	EMIOS_CSR[22]—Status register	R	0x0000_0000	28.3.2.9/28-19
0x02F4	EMIOS_ALTA[22]—Alternate A register	R/W	0x0000_0000	28.3.2.10/28-19
0x02F8–0x02FF	Reserved			
0x0300	EMIOS_CADR[23]—Channel A data register	R/W	0x0000_0000	28.3.2.5/28-12
0x0304	EMIOS_CBDR[23]—Channel B data register	R/W	0x0000_0000	28.3.2.6/28-12
0x0308	EMIOS_CCNTR[23]—Counter register	R	0x0000_0000	28.3.2.7/28-13
0x030C	EMIOS_CCR[23]—Control register	R/W	0x0000_0000	28.3.2.8/28-14
0x0310	EMIOS_CSR[23]—Status register	R	0x0000_0000	28.3.2.9/28-19
0x0314	EMIOS_ALTA[23]—Alternate A register	R/W	0x0000_0000	28.3.2.10/28-19
0x0318–0x031F	Reserved			
0x0320	EMIOS_CADR[24]—Channel A data register	R/W	0x0000_0000	28.3.2.5/28-12
0x0324	EMIOS_CBDR[24]—Channel B data register	R/W	0x0000_0000	28.3.2.6/28-12
0x0328	EMIOS_CCNTR[24]—Counter register	R	0x0000_0000	28.3.2.7/28-13

Table A-4. PXN20 Detailed Register Map (continued)

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
0x032C	EMIOS_CCR[24]—Control register	R/W	0x0000_0000	28.3.2.8/28-14
0x0330	EMIOS_CSR[24]—Status register	R	0x0000_0000	28.3.2.9/28-19
0x0334	EMIOS_ALTA[24]—Alternate A register	R/W	0x0000_0000	28.3.2.10/28-19
0x0338–0x033F	Reserved			
0x0340	EMIOS_CADR[25]—Channel A data register	R/W	0x0000_0000	28.3.2.5/28-12
0x0344	EMIOS_CBDR[25]—Channel B data register	R/W	0x0000_0000	28.3.2.6/28-12
0x0348	EMIOS_CCNTR[25]—Counter register	R	0x0000_0000	28.3.2.7/28-13
0x034C	EMIOS_CCR[25]—Control register	R/W	0x0000_0000	28.3.2.8/28-14
0x0350	EMIOS_CSR[25]—Status register	R	0x0000_0000	28.3.2.9/28-19
0x0354	EMIOS_ALTA[25]—Alternate A register	R/W	0x0000_0000	28.3.2.10/28-19
0x0358–0x035F	Reserved			
0x0360	EMIOS_CADR[26]—Channel A data register	R/W	0x0000_0000	28.3.2.5/28-12
0x0364	EMIOS_CBDR[26]—Channel B data register	R/W	0x0000_0000	28.3.2.6/28-12
0x0368	EMIOS_CCNTR[26]—Counter register	R	0x0000_0000	28.3.2.7/28-13
0x036C	EMIOS_CCR[26]—Control register	R/W	0x0000_0000	28.3.2.8/28-14
0x0370	EMIOS_CSR[26]—Status register	R	0x0000_0000	28.3.2.9/28-19
0x0374	EMIOS_ALTA[26]—Alternate A register	R/W	0x0000_0000	28.3.2.10/28-19
0x0378–0x037F	Reserved			
0x0380	EMIOS_CADR[27]—Channel A data register	R/W	0x0000_0000	28.3.2.5/28-12
0x0384	EMIOS_CBDR[27]—Channel B data register	R/W	0x0000_0000	28.3.2.6/28-12
0x0388	EMIOS_CCNTR[27]—Counter register	R	0x0000_0000	28.3.2.7/28-13
8C0x03	EMIOS_CCR[27]—Control register	R/W	0x0000_0000	28.3.2.8/28-14
0x0390	EMIOS_CSR[27]—Status register	R	0x0000_0000	28.3.2.9/28-19
0x0394	EMIOS_ALTA[27]—Alternate A register	R/W	0x0000_0000	28.3.2.10/28-19
0x0398–0x039F	Reserved			
0x03A0	EMIOS_CADR[28]—Channel A data register	R/W	0x0000_0000	28.3.2.5/28-12
0x03A4	EMIOS_CBDR[28]—Channel B data register	R/W	0x0000_0000	28.3.2.6/28-12
0x03A8	EMIOS_CCNTR[28]—Counter register	R	0x0000_0000	28.3.2.7/28-13
0x03AC	EMIOS_CCR[28]—Control register	R/W	0x0000_0000	28.3.2.8/28-14
0x03B0	EMIOS_CSR[28]—Status register	R	0x0000_0000	28.3.2.9/28-19
0x03B4	EMIOS_ALTA[28]—Alternate A register	R/W	0x0000_0000	28.3.2.10/28-19
0x03B8–0x03BF	Reserved			

Table A-4. PXN20 Detailed Register Map (continued)

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
0x03C0	EMIOS_CADR[29]—Channel A data register	R/W	0x0000_0000	28.3.2.5/28-12
0x03C4	EMIOS_CBDR[29]—Channel B data register	R/W	0x0000_0000	28.3.2.6/28-12
0x03C8	EMIOS_CCNTR[29]—Counter register	R	0x0000_0000	28.3.2.7/28-13
0x03CC	EMIOS_CCR[29]—Control register	R/W	0x0000_0000	28.3.2.8/28-14
0x03D0	EMIOS_CSR[29]—Status register	R	0x0000_0000	28.3.2.9/28-19
0x03D4	EMIOS_ALTA[29]—Alternate A register	R/W	0x0000_0000	28.3.2.10/28-19
0x03D8–0x03DF	Reserved			
0x03E0	EMIOS_CADR[30]—Channel A data register	R/W	0x0000_0000	28.3.2.5/28-12
0x03E4	EMIOS_CBDR[30]—Channel B data register	R/W	0x0000_0000	28.3.2.6/28-12
0x03E8	EMIOS_CCNTR[30]—Counter register	R	0x0000_0000	28.3.2.7/28-13
0x03EC	EMIOS_CCR[30]—Control register	R/W	0x0000_0000	28.3.2.8/28-14
0x03F0	EMIOS_CSR[30]—Status register	R	0x0000_0000	28.3.2.9/28-19
0x03F4	EMIOS_ALTA[30]—Alternate A register	R/W	0x0000_0000	28.3.2.10/28-19
0x03F8–0x03FF	Reserved			
0x0400	EMIOS_CADR[31]—Channel A data register	R/W	0x0000_0000	28.3.2.5/28-12
0x0404	EMIOS_CBDR[31]—Channel B data register	R/W	0x0000_0000	28.3.2.6/28-12
0x0408	EMIOS_CCNTR[31]—Counter register	R	0x0000_0000	28.3.2.7/28-13
0x040C	EMIOS_CCR[31]—Control register	R/W	0x0000_0000	28.3.2.8/28-14
0x0410	EMIOS_CSR[31]—Status register	R	0x0000_0000	28.3.2.9/28-19
0x0414	EMIOS_ALTA[31]—Alternate A register	R/W	0x0000_0000	28.3.2.10/28-19
0x0418–0x3FFF	Reserved			
0xFFFE_8000	**SIU** **Chapter 8, System Integration Unit (SIU)**			
0x0000–0x0003	Reserved			
0x0004	SIU_MIDR—MCU ID register	R	—[5]	8.3.2.1/8-13
0x0008–0x000B	Reserved			
0x000C	SIU_RSR—Reset status register	R/W	0x8000_000U	8.3.2.2/8-14
0x0010	SIU_SRCR—System reset control register	R/W	0x0800_C000	8.3.2.3/8-15
0x0014	SIU_EISR—SIU External Interrupt Status Register	R/W	0x0000_0000	8.3.2.4/8-16
0x0018	SIU_DIRER—DMA/interrupt request enable register	R/W	0x0000_0000	8.3.2.5/8-17
0x001C	SIU_DIRSR—DMA/interrupt request select register	R/W	0x0000_0000	8.3.2.6/8-18
0x0020	SIU_OSR—Overrun status register	R/W	0x0000_0000	8.3.2.7/8-19

Table A-4. PXN20 Detailed Register Map (continued)

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
0x0024	SIU_ORER—Overrun request enable register	R/W	0x0000_0000	8.3.2.8/8-19
0x0028	SIU_IREER—External IRQ rising-edge event enable register	R/W	0x0000_0000	8.3.2.9/8-20
0x002C	SIU_IFEER—External IRQ falling-edge event enable register	R/W	0x0000_0000	8.3.2.10/8-20
0x0030	SIU_IDFR—External IRQ digital filter register	R/W	0x0000_0000	8.3.2.11/8-21
0x0034	SIU_IFIR—External IRQ filtered input register	R/W	0x0000_0000	8.3.2.12/8-22
0x0038–0x003F	Reserved			
0x0040	SIU_PCR0 —Pad configuration register 0 (PA0)	R/W	0x0000	8.3.2.13/8-22
0x0042	SIU_PCR1—Pad configuration register 1 (PA1)	R/W	0x0000	8.3.2.13/8-22
0x0044	SIU_PCR2—Pad configuration register 2 (PA2)	R/W	0x0000	8.3.2.13/8-22
0x0046	SIU_PCR3—Pad configuration register 3 (PA3)	R/W	0x0000	8.3.2.13/8-22
0x0048	SIU_PCR4—Pad configuration register 4 (PA4)	R/W	0x0000	8.3.2.13/8-22
0x004A	SIU_PCR5—Pad configuration register 5 (PA5)	R/W	0x0000	8.3.2.13/8-22
0x004C	SIU_PCR6—Pad configuration register 6 (PA6)	R/W	0x0000	8.3.2.13/8-22
0x004E	SIU_PCR7—Pad configuration register 7 (PA7)	R/W	0x0000	8.3.2.13/8-22
0x0050	SIU_PCR8—Pad configuration register 8 (PA8)	R/W	0x0000	8.3.2.13/8-22
0x0052	SIU_PCR9—Pad configuration register 9 (PA9)	R/W	0x0000	8.3.2.13/8-22
0x0054	SIU_PCR10—Pad configuration register 10 (PA10)	R/W	0x0000	8.3.2.13/8-22
0x0056	SIU_PCR11—Pad configuration register 11 (PA11)	R/W	0x0000	8.3.2.13/8-22
0x0058	SIU_PCR12—Pad configuration register 12 (PA12)	R/W	0x0000	8.3.2.13/8-22
0x005A	SIU_PCR13—Pad configuration register 13 (PA13)	R/W	0x0000	8.3.2.13/8-22
0x005C	SIU_PCR14—Pad configuration register 14 (PA14)	R/W	0x0000	8.3.2.13/8-22
0x005E	SIU_PCR15—Pad configuration register 15 (PA15)	R/W	0x0000	8.3.2.13/8-22
0x0060	SIU_PCR16—Pad configuration register 16 (PB0)	R/W	0x0000	8.3.2.13/8-22
0x0062	SIU_PCR17—Pad configuration register 17 (PB1)	R/W	0x0000	8.3.2.13/8-22
0x0064	SIU_PCR18—Pad configuration register 18 (PB2)	R/W	0x0000	8.3.2.13/8-22
0x0066	SIU_PCR19—Pad configuration register 19 (PB3)	R/W	0x0000	8.3.2.13/8-22
0x0068	SIU_PCR20—Pad configuration register 20 (PB4)	R/W	0x0000	8.3.2.13/8-22
0x006A	SIU_PCR21—Pad configuration register 21 (PB5)	R/W	0x0000	8.3.2.13/8-22
0x006C	SIU_PCR22—Pad configuration register 22 (PB6)	R/W	0x0000	8.3.2.13/8-22
0x006E	SIU_PCR23—Pad configuration register 23 (PB7)	R/W	0x0000	8.3.2.13/8-22
0x0070	SIU_PCR24—Pad configuration register 24 (PB8)	R/W	0x0000	8.3.2.13/8-22
0x0072	SIU_PCR25—Pad configuration register 25 (PB9)	R/W	0x0000	8.3.2.13/8-22

Table A-4. PXN20 Detailed Register Map (continued)

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
0x0074	SIU_PCR26—Pad configuration register 26 (PB10)	R/W	0x0000	8.3.2.13/8-22
0x0076	SIU_PCR27—Pad configuration register 27 (PB11)	R/W	0x0000	8.3.2.13/8-22
0x0078	SIU_PCR28—Pad configuration register 28 (PB12)	R/W	0x0000	8.3.2.13/8-22
0x007A	SIU_PCR29—Pad configuration register 29 (PB13)	R/W	0x0000	8.3.2.13/8-22
0x007C	SIU_PCR30—Pad configuration register 30 (PB14)	R/W	0x0000	8.3.2.13/8-22
0x007E	SIU_PCR31—Pad configuration register 31 (PB15)	R/W	0x0000	8.3.2.13/8-22
0x0080	SIU_PCR32—Pad configuration register 32 (PC0)	R/W	0x0000	8.3.2.13/8-22
0x0082	SIU_PCR33—Pad configuration register 33 (PC1)	R/W	0x0000	8.3.2.13/8-22
0x0084	SIU_PCR34—Pad configuration register 34 (PC2)	R/W	0x0000	8.3.2.13/8-22
0x0086	SIU_PCR35—Pad configuration register 35 (PC3)	R/W	0x0000	8.3.2.13/8-22
0x0088	SIU_PCR36—Pad configuration register 36 (PC4)	R/W	0x0000	8.3.2.13/8-22
0x008A	SIU_PCR37—Pad configuration register 37 (PC5)	R/W	0x0000	8.3.2.13/8-22
0x008C	SIU_PCR38—Pad configuration register 38 (PC6)	R/W	0x0000	8.3.2.13/8-22
0x008E	SIU_PCR39—Pad configuration register 39 (PC7)	R/W	0x0000	8.3.2.13/8-22
0x0090	SIU_PCR40—Pad configuration register 40 (PC8)	R/W	0x0000	8.3.2.13/8-22
0x0092	SIU_PCR41—Pad configuration register 41 (PC9)	R/W	0x0000	8.3.2.13/8-22
0x0094	SIU_PCR42—Pad configuration register 42 (PC10)	R/W	0x0000	8.3.2.13/8-22
0x0096	SIU_PCR43—Pad configuration register 43 (PC11)	R/W	0x0000	8.3.2.13/8-22
0x0098	SIU_PCR44—Pad configuration register 44 (PC12)	R/W	0x0000	8.3.2.13/8-22
0x009A	SIU_PCR45—Pad configuration register 45 (PC13)	R/W	0x0000	8.3.2.13/8-22
0x009C	SIU_PCR46—Pad configuration register 46 (PC14)	R/W	0x0000	8.3.2.13/8-22
0x009E	SIU_PCR47—Pad configuration register 47 (PC15)	R/W	0x0000	8.3.2.13/8-22
0x00A0	SIU_PCR48—Pad configuration register 48 (PD0)	R/W	0x0000	8.3.2.13/8-22
0x00A2	SIU_PCR49—Pad configuration register 49 (PD1)	R/W	0x0000	8.3.2.13/8-22
0x00A4	SIU_PCR50—Pad configuration register 50 (PD2)	R/W	0x0000	8.3.2.13/8-22
0x00A6	SIU_PCR51—Pad configuration register 51 (PD3)	R/W	0x0000	8.3.2.13/8-22
0x00A8	SIU_PCR52—Pad configuration register 52 (PD4)	R/W	0x0000	8.3.2.13/8-22
0x00AA	SIU_PCR53—Pad configuration register 53 (PD5)	R/W	0x0000	8.3.2.13/8-22
0x00AC	SIU_PCR54—Pad configuration register 54 (PD6)	R/W	0x0000	8.3.2.13/8-22
0x00AE	SIU_PCR55—Pad configuration register 55 (PD7)	R/W	0x0000	8.3.2.13/8-22
0x00B0	SIU_PCR56—Pad configuration register 56 (PD8)	R/W	0x0000	8.3.2.13/8-22
0x00B2	SIU_PCR57—Pad configuration register 57 (PD9)	R/W	0x0000	8.3.2.13/8-22

Table A-4. PXN20 Detailed Register Map (continued)

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
0x00B4	SIU_PCR58—Pad configuration register 58 (PD10)	R/W	0x0000	8.3.2.13/8-22
0x00B6	SIU_PCR59—Pad configuration register 59 (PD11)	R/W	0x0000	8.3.2.13/8-22
0x00B8	SIU_PCR60—Pad configuration register 60 (PD12)	R/W	0x0000	8.3.2.13/8-22
0x00BA	SIU_PCR61—Pad configuration register 61 (PD13)	R/W	0x0000	8.3.2.13/8-22
0x00BC	SIU_PCR62—Pad configuration register 62 (PD14)	R/W	0x0000	8.3.2.13/8-22
0x00BE	SIU_PCR63—Pad configuration register 63 (PD15)	R/W	0x0000	8.3.2.13/8-22
0x00C0	SIU_PCR64—Pad configuration register 64 (PE0)	R/W	0x0000	8.3.2.13/8-22
0x00C2	SIU_PCR65—Pad configuration register 65 (PE1)	R/W	0x0000	8.3.2.13/8-22
0x00C4	SIU_PCR66—Pad configuration register 66 (PE2)	R/W	0x0000	8.3.2.13/8-22
0x00C6	SIU_PCR67—Pad configuration register 67 (PE3)	R/W	0x0000	8.3.2.13/8-22
0x00C8	SIU_PCR68—Pad configuration register 68 (PE4)	R/W	0x0000	8.3.2.13/8-22
0x00CA	SIU_PCR69—Pad configuration register 69 (PE5)	R/W	0x0000	8.3.2.13/8-22
0x00CC	SIU_PCR70—Pad configuration register 70 (PE6)	R/W	0x0000	8.3.2.13/8-22
0x00CE	SIU_PCR71—Pad configuration register 71 (PE7)	R/W	0x0000	8.3.2.13/8-22
0x00D0	SIU_PCR72—Pad configuration register 72 (PE8)	R/W	0x0000	8.3.2.13/8-22
0x00D2	SIU_PCR73—Pad configuration register 73 (PE9)	R/W	0x0000	8.3.2.13/8-22
0x00D4	SIU_PCR74—Pad configuration register 74 (PE10)	R/W	0x0000	8.3.2.13/8-22
0x00D6	SIU_PCR75—Pad configuration register 75 (PE11)	R/W	0x0000	8.3.2.13/8-22
0x00D8	SIU_PCR76—Pad configuration register 76 (PE12)	R/W	0x0000	8.3.2.13/8-22
0x00DA	SIU_PCR77—Pad configuration register 77 (PE13)	R/W	0x0000	8.3.2.13/8-22
0x00DC	SIU_PCR78—Pad configuration register 78 (PE14)	R/W	0x0000	8.3.2.13/8-22
0x00DE	SIU_PCR79—Pad configuration register 79 (PE15)	R/W	0x0000	8.3.2.13/8-22
0x00E0	SIU_PCR80—Pad configuration register 80 (PF0)	R/W	0x0000	8.3.2.13/8-22
0x00E2	SIU_PCR81—Pad configuration register 81 (PF1)	R/W	0x0000	8.3.2.13/8-22
0x00E4	SIU_PCR82—Pad configuration register 82 (PF2)	R/W	0x0000	8.3.2.13/8-22
0x00E6	SIU_PCR83—Pad configuration register 83 (PF3)	R/W	0x0000	8.3.2.13/8-22
0x00E8	SIU_PCR84—Pad configuration register 84 (PF4)	R/W	0x0000	8.3.2.13/8-22
0x00EA	SIU_PCR85—Pad configuration register 85 (PF5)	R/W	0x0000	8.3.2.13/8-22
0x00EC	SIU_PCR86—Pad configuration register 86 (PF6)	R/W	0x0000	8.3.2.13/8-22
0x00EE	SIU_PCR87—Pad configuration register 87 (PF7)	R/W	0x0000	8.3.2.13/8-22
0x00F0	SIU_PCR88—Pad configuration register 88 (PF8)	R/W	0x0000	8.3.2.13/8-22
0x00F2	SIU_PCR89—Pad configuration register 89 (PF9)	R/W	0x0000	8.3.2.13/8-22

Table A-4. PXN20 Detailed Register Map (continued)

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
0x00F4	SIU_PCR90—Pad configuration register 90 (PF10)	R/W	0x0000	8.3.2.13/8-22
0x00F6	SIU_PCR91—Pad configuration register 91 (PF11)	R/W	0x0000	8.3.2.13/8-22
0x00F8	SIU_PCR92—Pad configuration register 92 (PF12)	R/W	0x0000	8.3.2.13/8-22
0x00FA	SIU_PCR93—Pad configuration register 93 (PF13)	R/W	0x0000	8.3.2.13/8-22
0x00FC	SIU_PCR94—Pad configuration register 94 (PF14)	R/W	0x0000	8.3.2.13/8-22
0x00FE	SIU_PCR95—Pad configuration register 95 (PF15)	R/W	0x0000	8.3.2.13/8-22
0x0100	SIU_PCR96—Pad configuration register 96 (PG0)	R/W	0x0000	8.3.2.13/8-22
0x0102	SIU_PCR97—Pad configuration register 97 (PG1)	R/W	0x0000	8.3.2.13/8-22
0x0104	SIU_PCR98—Pad configuration register 98 (PG2)	R/W	0x0000	8.3.2.13/8-22
0x0106	SIU_PCR99—Pad configuration register 99 (PG3)	R/W	0x0000	8.3.2.13/8-22
0x0108	SIU_PCR100—Pad configuration register 100 (PG4)	R/W	0x0000	8.3.2.13/8-22
0x010A	SIU_PCR101—Pad configuration register 101 (PG5)	R/W	0x0000	8.3.2.13/8-22
0x010C	SIU_PCR102—Pad configuration register 102 (PG6)	R/W	0x0000	8.3.2.13/8-22
0x010E	SIU_PCR103—Pad configuration register 103 (PG7)	R/W	0x0000	8.3.2.13/8-22
0x0110	SIU_PCR104—Pad configuration register 104 (PG8)	R/W	0x0000	8.3.2.13/8-22
0x0112	SIU_PCR105—Pad configuration register 105 (PG9)	R/W	0x0000	8.3.2.13/8-22
0x0114	SIU_PCR106—Pad configuration register 106 (PG10)	R/W	0x0000	8.3.2.13/8-22
0x0116	SIU_PCR107—Pad configuration register 107 (PG11)	R/W	0x0000	8.3.2.13/8-22
0x0118	SIU_PCR108—Pad configuration register 108 (PG12)	R/W	0x0000	8.3.2.13/8-22
0x011A	SIU_PCR109—Pad configuration register 109 (PG13)	R/W	0x0000	8.3.2.13/8-22
0x011C	SIU_PCR110—Pad configuration register 110 (PG14)	R/W	0x0000	8.3.2.13/8-22
0x011E	SIU_PCR111—Pad configuration register 111 (PG15)	R/W	0x0000	8.3.2.13/8-22
0x0120	SIU_PCR112—Pad 4configuration register 112 (PH0)	R/W	0x0000	8.3.2.13/8-22
0x0122	SIU_PCR113—Pad configuration register 113 (PH1)	R/W	0x0000	8.3.2.13/8-22
0x0124	SIU_PCR114—Pad configuration register 114 (PH2)	R/W	0x0000	8.3.2.13/8-22
0x0126	SIU_PCR115—Pad configuration register 115 (PH3)	R/W	0x0000	8.3.2.13/8-22
0x0128	SIU_PCR116—Pad configuration register 116 (PH4)	R/W	0x0000	8.3.2.13/8-22
0x012A	SIU_PCR117—Pad configuration register 117 (PH5)	R/W	0x0000	8.3.2.13/8-22
0x012C	SIU_PCR118—Pad configuration register 118 (PH6)	R/W	0x0000	8.3.2.13/8-22
0x012E	SIU_PCR119—Pad configuration register 119 (PH7)	R/W	0x0000	8.3.2.13/8-22
0x0130	SIU_PCR120—Pad configuration register 120 (PH8)	R/W	0x0000	8.3.2.13/8-22
0x0132	SIU_PCR121—Pad configuration register 121 (PH9)	R/W	0x0000	8.3.2.13/8-22

Table A-4. PXN20 Detailed Register Map (continued)

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
0x0134	SIU_PCR122—Pad configuration register 122 (PH10)	R/W	0x0000	8.3.2.13/8-22
0x0136	SIU_PCR123—Pad configuration register 123 (PH11)	R/W	0x0000	8.3.2.13/8-22
0x0138	SIU_PCR124—Pad configuration register 124 (PH12)	R/W	0x0000	8.3.2.13/8-22
0x013A	SIU_PCR125—Pad configuration register 125 (PH13)	R/W	0x0000	8.3.2.13/8-22
0x013C	SIU_PCR126—Pad configuration register 126 (PH14)	R/W	0x0000	8.3.2.13/8-22
0x013E	SIU_PCR127—Pad configuration register 127 (PH15)	R/W	0x0000	8.3.2.13/8-22
0x0140	SIU_PCR128—Pad configuration register 128 (PJ0)	R/W	0x0000	8.3.2.13/8-22
0x0142	SIU_PCR129—Pad configuration register 129 (PJ1)	R/W	0x0000	8.3.2.13/8-22
0x0144	SIU_PCR130—Pad configuration register 130 (PJ2)	R/W	0x0000	8.3.2.13/8-22
0x0146	SIU_PCR131—Pad configuration register 131 (PJ3)	R/W	0x0000	8.3.2.13/8-22
0x0148	SIU_PCR132—Pad configuration register 132 (PJ4)	R/W	0x0000	8.3.2.13/8-22
0x014A	SIU_PCR133—Pad configuration register 133 (PJ5)	R/W	0x0000	8.3.2.13/8-22
0x014C	SIU_PCR134—Pad configuration register 134 (PJ6)	R/W	0x0000	8.3.2.13/8-22
0x014E	SIU_PCR135—Pad configuration register 135 (PJ7)	R/W	0x0000	8.3.2.13/8-22
0x0150	SIU_PCR136—Pad configuration register 136 (PJ8)	R/W	0x0000	8.3.2.13/8-22
0x0152	SIU_PCR137—Pad configuration register 137 (PJ9)	R/W	0x0000	8.3.2.13/8-22
0x0154	SIU_PCR138—Pad configuration register 138 (PJ10)	R/W	0x0000	8.3.2.13/8-22
0x0156	SIU_PCR139—Pad configuration register 139 (PJ11)	R/W	0x0000	8.3.2.13/8-22
0x0158	SIU_PCR140—Pad configuration register 140 (PJ12)	R/W	0x0000	8.3.2.13/8-22
0x015A	SIU_PCR141—Pad configuration register 141 (PJ13)	R/W	0x0000	8.3.2.13/8-22
0x015C	SIU_PCR142—Pad configuration register 142 (PJ14)	R/W	0x0000	8.3.2.13/8-22
0x015E	SIU_PCR143—Pad configuration register 143 (PJ15)	R/W	0x0000	8.3.2.13/8-22
0x0160	SIU_PCR144—Pad configuration register 144 (PK0)	R/W	0x0000	8.3.2.13/8-22
0x0162	SIU_PCR145—Pad configuration register 145 (PK1)	R/W	0x0000	8.3.2.13/8-22
0x0164	SIU_PCR146—Pad configuration register 146 (PK2)	R/W	0x0000	8.3.2.13/8-22
0x0166	SIU_PCR147—Pad configuration register 147 (PK3)	R/W	0x0000	8.3.2.13/8-22
0x0168	SIU_PCR148—Pad configuration register 148 (PK4)	R/W	0x0000	8.3.2.13/8-22
0x016A	SIU_PCR149—Pad configuration register 149 (PK5)	R/W	0x0000	8.3.2.13/8-22
0x016C	SIU_PCR150—Pad configuration register 150 (PK6)	R/W	0x0000	8.3.2.13/8-22
0x016E	SIU_PCR151—Pad configuration register 151 (PK7)	R/W	0x0000	8.3.2.13/8-22
0x0170	SIU_PCR152—Pad configuration register 152 (PK8)	R/W	0x0000	8.3.2.13/8-22
0x0172	SIU_PCR153—Pad configuration register 153 (PK9)	R/W	0x0101	8.3.2.13/8-22

Table A-4. PXN20 Detailed Register Map (continued)

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
0x0174	SIU_PCR154—Pad configuration register 154 (PK10)	R/W	0x0000	8.3.2.13/8-22
0x0176–0x060F	Reserved			
0x0610	SIU_GPDO16_19—GPIO pin data output register 16 – 19	R/W	0x0000_0000	8.3.2.14/8-26
0x0614	SIU_GPDO20_23—GPIO pin data output register 20 – 23	R/W	0x0000_0000	8.3.2.14/8-26
0x0618	SIU_GPDO24_27—GPIO pin data output register 24 – 27	R/W	0x0000_0000	8.3.2.14/8-26
0x061C	SIU_GPDO28_31—GPIO pin data output register 28 – 31	R/W	0x0000_0000	8.3.2.14/8-26
0x0620	SIU_GPDO32_35—GPIO pin data output register 32 – 35	R/W	0x0000_0000	8.3.2.14/8-26
0x0624	SIU_GPDO36_39—GPIO pin data output register 36 – 39	R/W	0x0000_0000	8.3.2.14/8-26
0x0628	SIU_GPDO40_43—GPIO pin data output register 40 – 43	R/W	0x0000_0000	8.3.2.14/8-26
0x062C	SIU_GPDO44_47—GPIO pin data output register 44 – 47	R/W	0x0000_0000	8.3.2.14/8-26
0x0630	SIU_GPDO48_51—GPIO pin data output register 48 – 51	R/W	0x0000_0000	8.3.2.14/8-26
0x0634	SIU_GPDO52_55—GPIO pin data output register 52 – 55	R/W	0x0000_0000	8.3.2.14/8-26
0x0638	SIU_GPDO56_59—GPIO pin data output register 56 – 59	R/W	0x0000_0000	8.3.2.14/8-26
0x063C	SIU_GPDO60_63—GPIO pin data output register 60 – 63	R/W	0x0000_0000	8.3.2.14/8-26
0x0640	SIU_GPDO64_67 —GPIO pin data output register 64 – 67	R/W	0x0000_0000	8.3.2.14/8-26
0x0644	SIU_GPDO68_71—GPIO pin data output register 68 – 71	R/W	0x0000_0000	8.3.2.14/8-26
0x0648	SIU_GPDO72_75—GPIO pin data output register 72 – 75	R/W	0x0000_0000	8.3.2.14/8-26
0x064C	SIU_GPDO76_79—GPIO pin data output register 76 – 79	R/W	0x0000_0000	8.3.2.14/8-26
0x0650	SIU_GPDO80_83—GPIO pin data output register 80 – 83	R/W	0x0000_0000	8.3.2.14/8-26
0x0654	SIU_GPDO84_87—GPIO pin data output register 84 – 87	R/W	0x0000_0000	8.3.2.14/8-26
0x0658	SIU_GPDO88_91—GPIO pin data output register 88 – 91	R/W	0x0000_0000	8.3.2.14/8-26
0x065C	SIU_GPDO92_95—GPIO pin data output register 92 – 95	R/W	0x0000_0000	8.3.2.14/8-26
0x0660	SIU_GPDO96_99—GPIO pin data output register 96 – 99	R/W	0x0000_0000	8.3.2.14/8-26
0x0664	SIU_GPDO100_103—GPIO pin data output register 100 – 103	R/W	0x0000_0000	8.3.2.14/8-26
0x0668	SIU_GPDO104_107—GPIO pin data output register 104 – 107	R/W	0x0000_0000	8.3.2.14/8-26
0x066C	SIU_GPDO108_111—GPIO pin data output register 108 – 111	R/W	0x0000_0000	8.3.2.14/8-26
0x0670	SIU_GPDO112_115—GPIO pin data output register 112 – 115	R/W	0x0000_0000	8.3.2.14/8-26
0x0674	SIU_GPDO116_119—GPIO pin data output register 116 – 119	R/W	0x0000_0000	8.3.2.14/8-26
0x0678	SIU_GPDO120_123—GPIO pin data output register 120 – 123	R/W	0x0000_0000	8.3.2.14/8-26
0x067C	SIU_GPDO124_127 —GPIO pin data output register 124 – 127	R/W	0x0000_0000	8.3.2.14/8-26
0x0680	SIU_GPDO128_131—GPIO pin data output register 128 – 131	R/W	0x0000_0000	8.3.2.14/8-26
0x0684	SIU_GPDO132_135—GPIO pin data output register 132 – 135	R/W	0x0000_0000	8.3.2.14/8-26

Table A-4. PXN20 Detailed Register Map (continued)

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
0x0688	SIU_GPDO136_139 —GPIO pin data output register 136 – 139	R/W	0x0000_0000	8.3.2.14/8-26
0x068C	SIU_GPDO140_143 —GPIO pin data output register 140 – 143	R/W	0x0000_0000	8.3.2.14/8-26
0x0690	SIU_GPDO144_147 —GPIO pin data output register 144 – 147	R/W	0x0000_0000	8.3.2.14/8-26
0x0694	SIU_GPDO148_151—GPIO pin data output register 148 – 151	R/W	0x0000_0000	8.3.2.14/8-26
0x0698	SIU_GPDO152_154 —GPIO pin data output register 152 – 154	R/W	0x0000_0000	8.3.2.14/8-26
0x0690–0x07FF	Reserved			
0x0800	SIU_GPDI0_3—GPIO pin data input register 0 – 3	R/W	0x0U0U_0U0U	8.3.2.15/8-28
0x0804	SIU_GPDI4_7—GPIO pin data input register 4 – 7	R/W	0x0U0U_0U0U	8.3.2.15/8-28
0x0808	SIU_GPDI8_11—GPIO pin data input register 8 – 11	R/W	0x0U0U_0U0U	8.3.2.15/8-28
0x080C	SIU_GPDI12_15—GPIO pin data input register 12 – 15	R/W	0x0U0U_0U0U	8.3.2.15/8-28
0x0810	SIU_GPDI16_19—GPIO pin data input register 16 – 19	R/W	0x0U0U_0U0U	8.3.2.15/8-28
0x0814	SIU_GPDI20_23—GPIO pin data input register 20 – 23	R/W	0x0U0U_0U0U	8.3.2.15/8-28
0x0818	SIU_GPDI24_27—GPIO pin data input register 24 – 27	R/W	0x0U0U_0U0U	8.3.2.15/8-28
0x081C	SIU_GPDI28_31—GPIO pin data input register 28 – 31	R/W	0x0U0U_0U0U	8.3.2.15/8-28
0x0820	SIU_GPDI32_35—GPIO pin data input register 32 – 35	R/W	0x0U0U_0U0U	8.3.2.15/8-28
0x0824	SIU_GPDI36_39—GPIO pin data input register 36 – 39	R/W	0x0U0U_0U0U	8.3.2.15/8-28
0x0828	SIU_GPDI40_43—GPIO pin data input register 40 – 43	R/W	0x0U0U_0U0U	8.3.2.15/8-28
0x082C	SIU_GPDI44_47—GPIO pin data input register 44 – 47	R/W	0x0U0U_0U0U	8.3.2.15/8-28
0x0830	SIU_GPDI48_51—GPIO pin data input register 48 – 51	R/W	0x0U0U_0U0U	8.3.2.15/8-28
0x0834	SIU_GPDI52_55—GPIO pin data input register 52 – 55	R/W	0x0U0U_0U0U	8.3.2.15/8-28
0x0838	SIU_GPDI56_59—GPIO pin data input register 56 – 59	R/W	0x0U0U_0U0U	8.3.2.15/8-28
0x083C	SIU_GPDI60_63—GPIO pin data input register 60 – 63	R/W	0x0U0U_0U0U	8.3.2.15/8-28
0x0840	SIU_GPDI64_67—GPIO pin data input register 64 – 67	R/W	0x0U0U_0U0U	8.3.2.15/8-28
0x0844	SIU_GPDI68_71—GPIO pin data input register 68 – 71	R/W	0x0U0U_0U0U	8.3.2.15/8-28
0x0848	SIU_GPDI72_75—GPIO pin data input register 72 – 75	R/W	0x0U0U_0U0U	8.3.2.15/8-28
0x084C	SIU_GPDI76_79—GPIO pin data input register 76 – 79	R/W	0x0U0U_0U0U	8.3.2.15/8-28
0x0850	SIU_GPDI80_83—GPIO pin data input register 80 – 83	R/W	0x0U0U_0U0U	8.3.2.15/8-28
0x0854	SIU_GPDI84_87—GPIO pin data input register 84 – 87	R/W	0x0U0U_0U0U	8.3.2.15/8-28
0x0858	SIU_GPDI88_91—GPIO pin data input register 88 – 91	R/W	0x0U0U_0U0U	8.3.2.15/8-28
0x085C	SIU_GPDI92_95—GPIO pin data input register 92 – 95	R/W	0x0U0U_0U0U	8.3.2.15/8-28
0x0860	SIU_GPDI96_99—GPIO pin data input register 96 – 99	R/W	0x0U0U_0U0U	8.3.2.15/8-28
0x0864	SIU_GPDI100_103—GPIO pin data input register 100 – 103	R/W	0x0U0U_0U0U	8.3.2.15/8-28

Table A-4. PXN20 Detailed Register Map (continued)

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
0x0868	SIU_GPDI104_107—GPIO pin data input register 104 – 107	R/W	0x0U0U_0U0U	8.3.2.15/8-28
0x086C	SIU_GPDI108_111—GPIO pin data input register 108 – 111	R/W	0x0U0U_0U0U	8.3.2.15/8-28
0x0870	SIU_GPDI112_115—GPIO pin data input register 112 – 115	R/W	0x0U0U_0U0U	8.3.2.15/8-28
0x0874	SIU_GPDI116_119—GPIO pin data input register 116 – 119	R/W	0x0U0U_0U0U	8.3.2.15/8-28
0x0878	SIU_GPDI120_123—GPIO pin data input register 120 – 123	R/W	0x0U0U_0U0U	8.3.2.15/8-28
0x087C	SIU_GPDI124_127—GPIO pin data input register 124 – 127	R/W	0x0U0U_0U0U	8.3.2.15/8-28
0x0880	SIU_GPDI128_131—GPIO pin data input register 128 – 131	R/W	0x0U0U_0U0U	8.3.2.15/8-28
0x0884	SIU_GPDI132_135—GPIO pin data input register 132 – 135	R/W	0x0U0U_0U0U	8.3.2.15/8-28
0x0888	SIU_GPDI136_139—GPIO pin data input register 136 – 139	R/W	0x0U0U_0U0U	8.3.2.15/8-28
0x088C	SIU_GPDI140_143—GPIO pin data input register 140 – 143	R/W	0x0U0U_0U0U	8.3.2.15/8-28
0x0890	SIU_GPDI144_147—GPIO pin data input register 144 – 147	R/W	0x0U0U_0U0U	8.3.2.15/8-28
0x0894	SIU_GPDI148_151—GPIO pin data input register 148 – 151	R/W	0x0U0U_0U0U	8.3.2.15/8-28
0x0898	SIU_GPDI152_154—GPIO pin data input register 152 – 154	R/W	0x0U0U_0U0U	8.3.2.15/8-28
0x089C–0x0903	Reserved			
0x0904	SIU_ISEL1—IMUX select register 1	R/W	0x0000_0000	8.3.2.16/8-29
0x0908	SIU_ISEL2—IMUX select register 2	R/W	0x0000_0000	8.3.2.17/8-33
0x090C–0x090F	Reserved			
0x0910	SIU_ISEL4—IMUX select register 4	R/W	0x0000_0000	8.3.2.18/8-35
0x0914–0x097F	Reserved			
0x0980	SIU_CCR—Chip configuration register	R/W	0x000U_0000	8.3.2.19/8-36
0x0984	SIU_ECCR—External clock control register	R/W	0x0000_1001	8.3.2.20/8-37
0x0988	SIU_GPR0—General purpose register 0	R/W	0x0000_0000	8.3.2.21/8-38
0x098C	SIU_GPR1—General purpose register 1	R/W	0x0000_0000	8.3.2.21/8-38
0x0990	SIU_GPR2—General purpose register 2	R/W	0x0000_0000	8.3.2.21/8-38
0x0994	SIU_GPR3—General purpose register 3	R/W	0x0000_0000	8.3.2.21/8-38
0x0998–0x099B	Reserved			
0x09A0	SIU_SYSCLK—System clock register	R/W	0x0000_0000	8.3.2.22/8-38
0x09A4	SIU_HLT0—Halt request 0	R/W	0x0000_0000	8.3.2.23/8-39
0x09A8	SIU_HLT1—Halt request 1	R/W	0x0000_0000	8.3.2.23/8-39
0x09AC	SIU_HLTACK0—Halt acknowledge 0	R/W	0x0000_0000	8.3.2.24/8-41
0x09B0	SIU_HLTACK1—Halt acknowledge 1	R/W	0x0000_0000	8.3.2.24/8-41
0x09B4	SIU_EMIOS_SEL0—eMIOS select register 0	R/W	0x0000_0000	8.3.2.25/8-44

Table A-4. PXN20 Detailed Register Map (continued)

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
0x09B8	SIU_EMIOS_SEL1—eMIOS select register 1	R/W	0x0000_0000	8.3.2.25/8-44
0x09BC	SIU_EMIOS_SEL2—eMIOS select register 2	R/W	0x0000_0000	8.3.2.25/8-44
0x09C0	SIU_EMIOS_SEL3—eMIOS select register 3	R/W	0x0000_0000	8.3.2.25/8-44
0x09C4	SIU_ISEL2A—External interrupt select register 2A	R/W	0x0000_0000	8.3.2.26/8-45
0x09C8–0x0BFF	Reserved			
0x0C00	SIU_PGPDO0—Parallel GPIO pin data output register 0	R/W	0x0000_0000	8.3.2.27/8-48
0x0C04	SIU_PGPDO1—Parallel GPIO pin data output register 1	R/W	0x0000_0000	8.3.2.28/8-48
0x0C08	SIU_PGPDO2—Parallel GPIO pin data output register 2	R/W	0x0000_0000	8.3.2.29/8-49
0x0C0C	SIU_PGPDO3—Parallel GPIO pin data output register 3	R/W	0x0000_0000	8.3.2.30/8-49
0x0C10	SIU_PGPDO4—Parallel GPIO pin data output register 4	R/W	0x0000_0000	8.3.2.31/8-49
0x0C14–0x0C3F	Reserved			
0x0C40	SIU_PGPDI0—Parallel GPIO pin data input register 0	R/W	—[3]	8.3.2.32/8-50
0x0C44	SIU_PGPDI1—Parallel GPIO pin data input register 1	R/W	—[3]	8.3.2.33/8-50
0x0C48	SIU_PGPDI2—Parallel GPIO pin data input register 2	R/W	—[3]	8.3.2.34/8-51
0x0C4C	SIU_PGPDI3—Parallel GPIO pin data input register 3	R/W	—[3]	8.3.2.35/8-51
0x0C50	SIU_PGPDI4—Parallel GPIO pin data input register 4	R/W	—[3]	8.3.2.36/8-52
0x0C54–0x0C83	Reserved			
0x0C84	SIU_MPGPDO1—Masked parallel GPIO data output register 1	R	0x0000_0000	8.3.2.37/8-52
0x0C88	SIU_MPGPDO2—Masked parallel GPIO data output register 2	R	0x0000_0000	8.3.2.38/8-53
0x0C8C	SIU_MPGPDO3—Masked parallel GPIO data output register 3	R	0x0000_0000	8.3.2.39/8-53
0x0C90	SIU_MPGPDO4—Masked parallel GPIO data output register 4	R	0x0000_0000	8.3.2.40/8-54
0x0C94	SIU_MPGPDO5—Masked parallel GPIO data output register 5	R	0x0000_0000	8.3.2.41/8-54
0x0C98	SIU_MPGPDO6—Masked parallel GPIO data output register 6	R	0x0000_0000	8.3.2.42/8-55
0x0C9C	SIU_MPGPDO7—Masked parallel GPIO data output register 7	R	0x0000_0000	8.3.2.43/8-55
0x0CA0	SIU_MPGPDO8—Masked parallel GPIO data output register 8	R	0x0000_0000	8.3.2.44/8-56
0x0CA4	SIU_MPGPDO9—Masked parallel GPIO data output register 9	R	0x0000_0000	8.3.2.45/8-56
0x0CA8–0x0CFF	Reserved			
0x0D00	SIU_DSPIAH—Masked serial GPO register for DSPI_A High	R/W	0x0000_0000	8.3.2.46/8-57
0x0D04	SIU_DSPIAL—Masked serial GPO register for DSPI_A Low	R/W	0x0000_0000	8.3.2.47/8-58
0x0D08	SIU_DSPIBH—Masked serial GPO register for DSPI_B High	R/W	0x0000_0000	8.3.2.48/8-58
0x0D0C	SIU_DSPIBL—Masked serial GPO register for DSPI_B Low	R/W	0x0000_0000	8.3.2.49/8-59
0x0D10	SIU_DSPICH—Masked serial GPO register for DSPI_C High	R/W	0x0000_0000	8.3.2.50/8-60

Table A-4. PXN20 Detailed Register Map (continued)

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
0x0D14	SIU_DSPICL—Masked serial GPO register for DSPI_C Low	R/W	0x0000_0000	8.3.2.51/8-60
0x0D18	SIU_DSPIDH—Masked serial GPO register for DSPI_D High	R/W	0x0000_0000	8.3.2.52/8-61
0x0D1C	SIU_DSPIDL—Masked serial GPO register for DSPI_D Low	R/W	0x0000_0000	8.3.2.53/8-62
0x0D20–0x0D43	Reserved			
0x0D44	SIU_EMIOSA—eMIOS select register for DSPI_A	R/W	0x0000_0000	8.3.2.54/8-62
0x0D48	SIU_DSPIAHLA—SIU_DSPIAH/L select register for DSPI_A	R/W	0x0000_0000	8.3.2.55/8-63
0x0D4C–0x0D53	Reserved			
0x0D54	SIU_EMIOSB—eMIOS select register for DSPI_B	R/W	0x0000_0000	8.3.2.56/8-64
0x0D58	SIU_DSPIBHLB—SIU_DSPIBH/L select register for DSPI_B	R/W	0x0000_0000	8.3.2.57/8-64
0x0D5C–0x0D63	Reserved			
0x0D64	SIU_EMIOSC—eMIOS select register for DSPI_C	R/W	0x0000_0000	8.3.2.58/8-65
0x0D68	SIU_DSPICHLC—H/L select register for DSPI_C	R/W	0x0000_0000	8.3.2.59/8-65
0x0D6C–0x0D73	Reserved			
0x0D74	SIU_EMIOSD—eMIOS select register for DSPI_D	R/W	0x0000_0000	8.3.2.60/8-66
0x0D78–0x0D7B	SIU_DSPIDHLD—SIU_DSPIDH/L select register for DSPI_D	R/W	0x0000_0000	8.3.2.61/8-67
0x0D7C–0x3FFF	Reserved			
0xFFFE_C000	**CRP** **Chapter 6, Clocks, Reset, and Power (CRP)**			
0x0000	CRP_CLKSRC—Clock source register	R/W	0x0001_1F3F	6.2.2.1/6-5
0x0004–0x000F	Reserved			
0x0010	CRP_RTCC—RTC control register	R/W	0x0000_0000	6.2.2.2/6-6
0x0014	CRP_RTSC—RTC status register	R	0x0000_0000	6.2.2.3/6-8
0x0018	CRP_RTCCNT—RTC counter register	R	0x0000_0000	6.2.2.4/6-9
0x001C–0x003F	Reserved			
0x0040	CRP_PWKENH—Pin wakeup enable high register	R/W	0x0000_0000	6.2.2.5/6-9
0x0044	CRP_PWKENL—Pin Wakeup enable low register	R/W	0x0000_0000	6.2.2.5/6-9
0x0048	CRP_PWKSRCIE—Pin wakeup source interrupt enable register	R/W	0x0000_0000	6.2.2.6/6-11
0x004C	CRP_PWKSRCF—Pin wakeup source flag register	R	0x0000_0000	6.2.2.7/6-11
0x0050	CRP_Z6VEC—Z6 reset vector register	R/W	0xFFFF_0001	6.2.2.8/6-12
0x0054	CRP_Z0VEC—Z0 reset vector register	R/W	0xFFFF_FFFE	6.2.2.9/6-12
0x0058	CRP_RECPTR—Recovery pointer register	R/W	0xFFFF_FFFC	6.2.2.10/6-13
0x005C–0x005F	Reserved			

Table A-4. PXN20 Detailed Register Map (continued)

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
0x0060	CRP_PSCR—Power status and control register	R/W	0x0000_0000	6.2.2.11/6-14
0x0064–0x006F	Reserved			
0x0070	CRP_SOCSC—SoC status and control register	R/W	0x4000_0020	6.2.2.12/6-15
0x0074–0x03FF	Reserved			
0xFFFF_0000	**PLL** **Chapter 7, Frequency Modulated Phase-Locked Loop (FMPLL)**			
0x0000	Reserved			
0x0004	SYNSR—FMPLL synthesizer status register	R/W	—[6]	7.3.2.1/7-3
0x0008	ESYNCR1—FMPLL enhanced synthesizer control register 1	R/W	0x8000_0030	7.3.2.2/7-5
0x000C	ESYNCR2—FMPLL enhanced synthesizer control register 2	R/W	0x0000_0003	7.3.2.3/7-7
0x0010–0x3FFF	Reserved			
0xFFFF_4000 – 0xFFFF_7FFF	**Reserved**			
0xFFFF_8000	**PFlash Configuration** **Chapter 12, Flash Memory Array and Control**			
0x0000	MCR—Module configuration register	R/W	0x0540_0600	12.3.2.1/12-6
0x0004	LML—Low-/Mid-address space block locking register	R/W	0x0013_03FF	12.3.2.2/12-10
0x0008	HBL—High-address space block locking register	R/W	0x0000_003F	12.3.2.3/12-12
0x000C	SLL—Secondary low-/mid-address space block locking register	R/W	0x0013_03FF	12.3.2.4/12-13
0x0010	LMS—Low-/mid-address space block select register	R/W	0x0000_0000	12.3.2.5/12-14
0x0014	HBS—High-address space block select register	R/W	0x0000_0000	12.3.2.6/12-15
0x0018	ADR—Address register	R/W	0x0000_0000	12.3.2.7/12-16
0x001C	PFCRP0—Platform flash configuration register for port 0	R/W	0x0800_FF00	12.3.2.8/12-17
0x0020	PFCRP1—Platform flash configuration register for port 1	R/W	0x3000_FF00	12.3.2.8/12-17
0x0024	PFAPR—Platform flash access protection register	R/W	0x00FF_FE00	12.3.2.9/12-20
0x0028	PFSACC—Platform flash supervisor access control register	R/W	0x00FF_FE08	12.3.2.10/12-21
0x002C	PFDACC—Platform flash data access control register	R/W	0x00FF_FE10	12.3.2.11/12-23
0x0030 – 0x0038	Reserved			
0x003C	UT0—UTest register 0	R/W	0x0000_0001	12.3.2.12/12-23
0x0040	UT0—UTest register 1	R/W	0x0000_0000	12.3.2.13/12-25
0x0044	UT0—UTest register 2	R/W	0x0000_0000	12.3.2.14/12-26
0x0048	UM0—User multiple input signature register 0	R/W	0x0000_0000	12.3.2.15/12-26

Table A-4. PXN20 Detailed Register Map (continued)

Address Offset from Module Base	Register	Access[1]	Reset Value[2]	Section/Page
0x004C	UM1—User multiple input signature register 1	R/W	0x0000_0000	12.3.2.15/12-26
0x0050	UM2—User multiple input signature register 2	R/W	0x0000_0000	12.3.2.15/12-26
0x0054	UM3—User multiple input signature register 3	R/W	0x0000_0000	12.3.2.15/12-26
0x0058	UM4—User multiple input signature register 4	R/W	0x0000_0000	12.3.2.15/12-26
0x0048 – 0x3FFF	Reserved			
0xFFFF_C000	**BAM** **Chapter 9, Boot Assist Module (BAM)**			
0x3FFC	BAM reset vector—first executed address after the reset	R	—[3]	9.3.1/9-3
0x3000	BAM start address	R	—[3]	9.3.1/9-3
0x3004	Internal boot start address	R	—[3]	9.3.1/9-3
0x3008	External boot start address	R	—[3]	9.3.1/9-3
0x300C	Parallel loader/serial boot start address	R	—[3]	9.3.1/9-3

1 In this column, R/W indicates a read/write register, R indicates a read-only register, and W indicates a write-only register. Note that R/W registers may contain some read-only or write-only bits.

2 In this column, the symbol “U” indicates one or more bits in a byte are undefined at reset. See the associated description for more information.

3 Reset value is indeterminate.

4 Some bits are read-only.

5 See register description for reset value.

6 See specific register description.

A.3 e200z6 Core SPR Numbers

Table A-5. e200z6 Core SPR Numbers (Supervisor Mode)

Register	Description	SPR (decimal)
General Registers		
XER	Integer Exception Register	1
LR	Link Register	8
CTR	Count Register	9
GPR0–GPR31	General Purpose Registers	N/A
Special Purpose Registers		
SPRG0	Special Purpose Register 0	272
SPRG1	Special Purpose Register 1	273
SPRG2	Special Purpose Register 2	274
SPRG3	Special Purpose Register 3	275

Table A-5. e200z6 Core SPR Numbers (Supervisor Mode) (continued)

Register	Description	SPR (decimal)
SPRG4	Special Purpose Register 4	276
SPRG5	Special Purpose Register 5	277
SPRG6	Special Purpose Register 6	278
SPRG7	Special Purpose Register 7	279
USPRG0	User Special Purpose Register	256
BUCSR	Branch Unit Control and Status Register	1013
Exception Handling/Control Registers		
SRR0	Save and Restore Register 0	26
SRR1	Save and Restore Register 1	27
CSRR0	Critical Save and Restore Register 0	58
CSRR1	Critical Save and Restore Register 1	59
DSRR0	Debug Save and Restore Register 0	574
DSRR1	Debug Save and Restore Register 1	575
ESR	Exception Syndrome Register	62
MCSR	Machine Check Syndrome Register	572
DEAR	Data Exception Address Register	61
IVPR	Interrupt Vector Prefix Register	63
IVOR1	Interrupt Vector Offset Register 1	401
IVOR2	Interrupt Vector Offset Register 2	402
IVOR3	Interrupt Vector Offset Register 3	403
IVOR4	Interrupt Vector Offset Register 4	404
IVOR5	Interrupt Vector Offset Register 5	405
IVOR6	Interrupt Vector Offset Register 6	406
IVOR7	Interrupt Vector Offset Register 7	407
IVOR8	Interrupt Vector Offset Register 8	408
IVOR9	Not Supported	—
IVOR10	Interrupt Vector Offset Register 10	410
IVOR11	Interrupt Vector Offset Register 11	411
IVOR12	Interrupt Vector Offset Register 12	412
IVOR13	Interrupt Vector Offset Register 13	413
IVOR14	Interrupt Vector Offset Register 14	414
IVOR15	Interrupt Vector Offset Register 15	415
IVOR32	Interrupt Vector Offset Register 32	528

Table A-5. e200z6 Core SPR Numbers (Supervisor Mode) (continued)

Register	Description	SPR (decimal)
IVOR33	Interrupt Vector Offset Register 33	529
IVOR34	Interrupt Vector Offset Register 34	530
Processor Control Registers		
MSR	Machine State Register	N/A
PVR	Processor Version Register	287
PIR	Processor ID Register	286
SVR	System Version Register	1023
HID0	Hardware Implementation Dependent Register 0	1008
HID1	Hardware Implementation Dependent Register 1	1009
Timer Registers		
TBL	Time Base Lower Register	284
TBU	Time Base Upper Register	285
TCR	Timer Control Register	340
TSR	Timer Status Register	336
DEC	Decrementer Register	22
DECAR	Decrementer Auto-reload Register	54
Debug Registers		
DBCR0	Debug Control Register 0	308
DBCR1	Debug Control Register 1	309
DBCR2	Debug Control Register 2	310
DBCR3	Debug Control Register 3	561
DBSR	Debug Status Register	304
DBCNT	Debug Counter Register	562
IAC1	Instruction Address Compare Register 1	312
IAC2	Instruction Address Compare Register 2	313
IAC3	Instruction Address Compare Register 3	314
IAC4	Instruction Address Compare Register 4	315
DAC1	Data Address Compare Register 1	316
DAC2	Data Address Compare Register 2	317
Memory Management Registers		
MAS0	MMU Assist Register 0	624
MAS1	MMU Assist Register 1	625
MAS2	MMU Assist Register 2r	626

Table A-5. e200z6 Core SPR Numbers (Supervisor Mode) (continued)

Register	Description	SPR (decimal)
MAS3	MMU Assist Register 3	627
MAS4	MMU Assist Register 4	628
MAS6	MMU Assist Register 6	630
PID0	Process ID Register	48
MMUCSR0	MMU Control and Status Register 0	1012
MMUCFG	MMU Configuration Register	1015
TLB0CFG	TLB 0 Configuration Register	688
TLB1CFG	TLB 1 Configuration Register	689
Cache Registers		
L1CFG0	L1 Cache Configuration Register	515
L1CSR0	L1 Cache Control and Status Register 0	1010
L1FINV0	L1 Cache Flush and Invalidate Control Register 0	1016
APU Registers		
SPEFSCR	SPE APU Status and Control Register	512

Table A-6. e200z6 Core SPR Numbers (User Mode)

Register	Description	SPR (decimal)
General Registers		
CTR	Count Register	9
LR	Link Register	8
XER	Integer Exception Register	1
GPR0–GPR31	General Purpose Registers	N/A
Special Purpose Registers		
SPRG4	Special Purpose Register 4	260
SPRG5	Special Purpose Register 5	261
SPRG6	Special Purpose Register 6	262
SPRG7	Special Purpose Register 7	263
USPRG0	User Special Purpose Register	256
Timer Registers		
TBL	Time Base Lower Register	268
TBU	Time Base Upper Register	269
Cache Registers		
L1CFG0	L1 Cache Configuration Register	515

Table A-6. e200z6 Core SPR Numbers (User Mode) (continued)

Register	Description	SPR (decimal)
APU Registers		
SPEFSCR	SPE APU Status and Control Register	512