

# MCF5275 Reference Manual

## Devices Supported:

MCF5274

MCF5274L

MCF5275

MCF5275L

Document Number: MCF5275RM

Rev. 2

07/2006

## ***How to Reach Us:***

### **Home Page:**

[www.freescale.com](http://www.freescale.com)

### **E-mail:**

[support@freescale.com](mailto:support@freescale.com)

### **USA/Europe or Locations Not Listed:**

Freescale Semiconductor  
Technical Information Center, CH370  
1300 N. Alma School Road  
Chandler, Arizona 85224  
+1-800-521-6274 or +1-480-768-2130  
[support@freescale.com](mailto:support@freescale.com)

### **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)  
[support@freescale.com](mailto:support@freescale.com)

### **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](mailto:support.japan@freescale.com)

### **Asia/Pacific:**

Freescale Semiconductor Hong Kong Ltd.  
Technical Information Center  
2 Dai King Street  
Tai Po Industrial Estate  
Tai Po, N.T., Hong Kong  
+800 26668334  
[support.asia@freescale.com](mailto: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 303-675-2140  
Fax: 303-675-2150  
[LDCForFreescaleSemiconductor@hibbertgroup.com](mailto:LDCForFreescaleSemiconductor@hibbertgroup.com)

Information in this document is provided solely to enable system and software implementers to use Freescale Semiconductor products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits or integrated circuits based on the information in this document.

Freescale Semiconductor reserves the right to make changes without further notice to any products herein. Freescale Semiconductor makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale Semiconductor assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters that may be provided in Freescale Semiconductor data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including "Typicals", must be validated for each customer application by customer's technical experts. Freescale Semiconductor does not convey any license under its patent rights nor the rights of others. Freescale Semiconductor products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Freescale Semiconductor product could create a situation where personal injury or death may occur. Should Buyer purchase or use Freescale Semiconductor products for any such unintended or unauthorized application, Buyer shall indemnify and hold Freescale Semiconductor and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Freescale Semiconductor was negligent regarding the design or manufacture of the part.



Freescale™ and the Freescale logo are trademarks of Freescale Semiconductor, Inc. All other product or service names are the property of their respective owners. © Freescale Semiconductor, Inc. 2006. All rights reserved.

MCF5275RM  
Rev. 2  
07/2006

Overview	1
Signal Descriptions	2
ColdFire Core	3
Enhanced Multiply-Accumulate Unit (EMAC)	4
Cache	5
Static RAM (SRAM)	6
Clock Module	7
Power Management	8
Chip Configuration Module (CCM)	9
Reset Controller Module	10
System Control Module (SCM)	11
General Purpose I/O Module	12
Interrupt Controller Modules	13
Edge Port Module (EPORT)	14
Chip Select Module	15
External Interface Module (EIM)	16
Synchronous DRAM Controller	17
DMA Controller Module	18
Fast Ethernet Controller (FEC)	19
Universal Serial Bus Device (USB)	20
Watchdog Timer Module	21
PWM Module	22
Programmable Interrupt Timers (PITs)	23
DMA Timers	24
Queued Serial Peripheral Interface (QSPI)	25
UART Modules	26
I <sup>2</sup> C interface	27
Message Digest Hardware Accelerator (MDHA)	28
Random Number Generator (RNG)	29
Symmetric Key Hardware Accelerator (SKHA)	30
IEEE 1149.1 Test Access Port (JTAG)	31
Debug Support	32
Register Memory Map Quick Reference	A
Index	IND

1	Overview
2	Signal Descriptions
3	ColdFire Core
4	Enhanced Multiply-Accumulate Unit (EMAC)
5	Cache
6	Static RAM (SRAM)
7	Clock Module
8	Power Management
9	Chip Configuration Module (CCM)
10	Reset Controller Module
11	System Control Module (SCM)
12	General Purpose I/O Module
13	Interrupt Controller Modules
14	Edge Port Module (EPORT)
15	Chip Select Module
16	External Interface Module (EIM)
17	Synchronous DRAM Controller
18	DMA Controller Module
19	Fast Ethernet Controller (FEC)
20	Universal Serial Bus Device (USB)
21	Watchdog Timer Module
22	PWM Module
23	Programmable Interrupt Timers (PITs)
24	DMA Timers
25	Queued Serial Peripheral Interface (QSPI)
26	UART Modules
27	I <sup>2</sup> C interface
28	Message Digest Hardware Accelerator (MDHA)
29	Random Number Generator (RNG)
30	Symmetric Key Hardware Accelerator (SKHA)
31	IEEE 1149.1 Test Access Port (JTAG)
32	Debug Support
A	Register Memory Map Quick Reference
IND	Index



# Contents

Paragraph Number	Title	Page Number
------------------	-------	-------------

## Chapter 1 Overview

1.1	MCF5275 Family Configurations.....	1-1
1.2	Block Diagram.....	1-2
1.3	Features.....	1-4
1.3.1	Feature Overview.....	1-4
1.3.2	V2 Core Overview.....	1-8
1.3.3	Integrated Debug Module.....	1-8
1.3.4	JTAG.....	1-9
1.3.5	On-chip Memories.....	1-9
1.3.5.1	Cache.....	1-9
1.3.5.2	SRAM.....	1-10
1.3.6	Fast Ethernet Controller (FEC).....	1-10
1.3.7	Universal Serial Bus (USB).....	1-10
1.3.8	UARTs.....	1-10
1.3.9	I <sup>2</sup> C Bus.....	1-10
1.3.10	QSPI.....	1-11
1.3.11	Cryptography.....	1-11
1.3.12	DMA Timers (DTIM0-DTIM3).....	1-11
1.3.13	Pulse Width Modulation (PWM) Module.....	1-11
1.3.14	Periodic Interrupt Timers (PIT0-PIT3).....	1-11
1.3.15	Software Watchdog Timer.....	1-12
1.3.16	Clock Module and Phase Locked Loop (PLL).....	1-12
1.3.17	Interrupt Controllers (INTC0, INTC1).....	1-12
1.3.18	DMA Controller.....	1-12
1.3.19	External Interface Module (EIM).....	1-12
1.3.20	Double Data Rate (DDR) Synchronous DRAM (SDRAM) Controller.....	1-13
1.3.21	Reset.....	1-13
1.3.22	GPIO.....	1-13
1.4	Documentation.....	1-14

## Chapter 2 Signal Descriptions

2.1	Introduction.....	2-1
2.1.1	Overview.....	2-1
2.2	Signal Properties Summary.....	2-3

# Contents

Paragraph Number	Title	Page Number
2.3	External Signal Descriptions .....	2-8
2.3.1	Reset Signals.....	2-8
2.3.2	PLL and Clock Signals .....	2-9
2.3.3	Mode Selection .....	2-9
2.3.4	External Memory Interface Signals .....	2-9
2.3.5	DDR SDRAM Controller Signals.....	2-10
2.3.6	External Interrupt Signals .....	2-11
2.3.7	Fast Ethernet Controller Signals .....	2-11
2.3.8	Queued Serial Peripheral Interface (QSPI).....	2-12
2.3.9	I2C I/O SIGNALS .....	2-13
2.3.10	UART Module Signals .....	2-13
2.3.11	USB Signals.....	2-14
2.3.12	DMA Timer Signals.....	2-15
2.3.13	Pulse Width Modulator Signals .....	2-15
2.3.14	Debug Support Signals .....	2-16
2.3.15	Test Signals.....	2-17
2.3.16	Power and Ground Pins .....	2-17
2.4	External Boot Mode.....	2-18

## Chapter 3 ColdFire Core

3.1	Processor Pipelines .....	3-1
3.2	Processor Register Description .....	3-2
3.2.1	User Programming Model .....	3-2
3.2.1.1	Data Registers (D0–D7) .....	3-2
3.2.1.2	Address Registers (A0–A6).....	3-3
3.2.1.3	Stack Pointer (A7) .....	3-3
3.2.1.4	Program Counter (PC) .....	3-3
3.2.1.5	Condition Code Register (CCR).....	3-4
3.2.2	EMAC Register Description.....	3-4
3.2.3	Supervisor Register Description .....	3-5
3.2.3.1	Status Register (SR).....	3-6
3.2.3.2	Supervisor/User Stack Pointers (A7 and OTHER_A7).....	3-7
3.2.3.3	Vector Base Register (VBR) .....	3-7
3.2.3.4	Cache Control Register (CACR) .....	3-7
3.2.3.5	Access Control Registers (ACR0, ACR1).....	3-8
3.2.3.6	SRAM Base Address Register (RAMBAR).....	3-8
3.3	Memory Map/Register Definition .....	3-8
3.4	Additions to the Instruction Set Architecture .....	3-9
3.5	Exception Processing Overview .....	3-9

# Contents

Paragraph Number	Title	Page Number
3.6	Exception Stack Frame Definition.....	3-11
3.7	Processor Exceptions .....	3-13
3.7.1	Access Error Exception .....	3-13
3.7.2	Address Error Exception.....	3-13
3.7.3	Illegal Instruction Exception.....	3-13
3.7.4	Divide-By-Zero.....	3-14
3.7.5	Privilege Violation.....	3-14
3.7.6	Trace Exception .....	3-14
3.7.7	Unimplemented Line-A Opcode.....	3-15
3.7.8	Unimplemented Line-F Opcode .....	3-15
3.7.9	Debug Interrupt.....	3-15
3.7.10	RTE and Format Error Exception.....	3-15
3.7.11	TRAP Instruction Exception.....	3-15
3.7.12	Interrupt Exception .....	3-15
3.7.13	Fault-on-Fault Halt .....	3-16
3.7.14	Reset Exception .....	3-16
3.8	Instruction Execution Timing .....	3-19
3.8.1	Timing Assumptions.....	3-19
3.8.2	MOVE Instruction Execution Times .....	3-20
3.9	Standard One Operand Instruction Execution Times .....	3-21
3.10	Standard Two Operand Instruction Execution Times.....	3-22
3.11	Miscellaneous Instruction Execution Times.....	3-24
3.12	EMAC Instruction Execution Times .....	3-25
3.13	Branch Instruction Execution Times .....	3-26
3.14	ColdFire Instruction Set Architecture Enhancements .....	3-26

## Chapter 4 Enhanced Multiply-Accumulate Unit (EMAC)

4.1	Multiply-Accumulate Unit.....	4-1
4.2	Introduction to the MAC.....	4-2
4.3	General Operation.....	4-3
4.4	Memory Map/Register Definition .....	4-6
4.4.1	MAC Status Register (MACSR).....	4-6
4.4.1.1	Fractional Operation Mode.....	4-9
4.4.2	Mask Register (MASK).....	4-11
4.5	EMAC Instruction Set Summary .....	4-12
4.5.1	EMAC Instruction Execution Times .....	4-13
4.5.2	Data Representation.....	4-14
4.5.3	MAC Opcodes .....	4-14

# Contents

Paragraph Number	Title	Page Number
<b>Chapter 5</b>		
<b>Cache</b>		
5.1	Introduction.....	5-1
5.1.1	Features.....	5-1
5.1.2	Physical Organization.....	5-1
5.1.3	Operation .....	5-3
5.1.3.1	Interaction with Other Modules.....	5-3
5.1.3.2	Memory Reference Attributes .....	5-4
5.1.3.3	Cache Coherency and Invalidation.....	5-4
5.1.3.4	Reset .....	5-5
5.1.3.5	Cache Miss Fetch Algorithm/Line Fills .....	5-5
5.2	Memory Map/Register Definition .....	5-6
5.2.1	Registers Description.....	5-7
5.2.1.1	Cache Control Register (CACR) .....	5-7
5.2.1.2	Access Control Registers (ACR0, ACR1).....	5-10

## **Chapter 6**

### **Static RAM (SRAM)**

6.1	Introduction.....	6-1
6.1.1	Features.....	6-1
6.1.2	Operation .....	6-1
6.2	Register Description .....	6-1
6.2.1	SRAM Base Address Register (RAMBAR).....	6-2
6.2.2	SRAM Initialization.....	6-4
6.2.3	SRAM Initialization Code .....	6-4
6.2.4	Power Management .....	6-5

## **Chapter 7**

### **Clock Module**

7.1	Introduction.....	7-1
7.1.1	Block Diagram.....	7-2
7.1.2	Features.....	7-4
7.1.3	Modes of Operation .....	7-4
7.1.3.1	Normal PLL Mode with Crystal Reference.....	7-5
7.1.3.2	Normal PLL Mode with External Reference.....	7-5
7.1.3.3	1:1 PLL Mode.....	7-5
7.1.3.4	External Clock Mode (Bypass Mode) .....	7-5

# Contents

Paragraph Number	Title	Page Number
7.1.3.5	Low-power Mode Operation .....	7-6
7.2	External Signal Descriptions .....	7-6
7.2.1	EXTAL .....	7-7
7.2.2	XTAL .....	7-7
7.2.3	CLKOUT .....	7-7
7.2.4	CLKMOD[1:0] .....	7-7
7.2.5	RSTOUT .....	7-7
7.3	Memory Map/Register Definition .....	7-8
7.3.1	Register Descriptions .....	7-8
7.3.1.1	Synthesizer Control Register (SYNCR) .....	7-8
7.3.1.2	Synthesizer Status Register (SYNSR) .....	7-11
7.4	Functional Description .....	7-13
7.4.1	System Clock Modes .....	7-13
7.4.2	Clock Operation During Reset .....	7-14
7.4.2.1	Power-On Reset (POR) .....	7-14
7.4.2.2	External Reset .....	7-15
7.4.3	System Clock Generation .....	7-15
7.4.4	Programming the Frequency Modulation .....	7-16
7.4.5	Frequency Modulation Depth Calibration .....	7-18
7.4.6	PLL Operation .....	7-21
7.4.6.1	Phase and Frequency Detector (PFD) .....	7-22
7.4.6.2	Charge Pump/Loop Filter .....	7-23
7.4.6.3	Current Controlled Oscillator (ICO) .....	7-23
7.4.6.4	Multiplication Factor Divider (MFD) .....	7-23
7.4.6.5	PLL Lock Detection .....	7-23
7.4.6.6	PLL Loss-of-Lock Conditions .....	7-24
7.4.6.7	PLL Loss-of-Lock Reset .....	7-25
7.4.6.8	PLL Loss-of-Lock Interrupt Request .....	7-25
7.4.6.9	Loss-of-Clock Detection .....	7-25
7.4.6.10	Loss-of-Clock Reset .....	7-25
7.4.6.11	Loss-of-Clock Interrupt Request .....	7-26
7.4.6.12	Alternate Clock Selection .....	7-26
7.4.6.13	Loss-of-Clock in Stop Mode .....	7-26
7.5	Interrupts .....	7-30

## Chapter 8 Power Management

8.1	Introduction .....	8-1
8.1.1	Features .....	8-1
8.2	Memory Map/Register Definition .....	8-1

# Contents

Paragraph Number	Title	Page Number
8.2.1	Register Descriptions.....	8-1
8.2.1.1	Low-Power Interrupt Control Register (LPICR).....	8-2
8.2.1.2	Low-Power Control Register (LPCR) .....	8-3
8.3	Functional Description.....	8-4
8.3.1	Low-Power Modes.....	8-4
8.3.1.1	Run Mode .....	8-5
8.3.1.2	Wait Mode .....	8-5
8.3.1.3	Doze Mode.....	8-5
8.3.1.4	Stop Mode.....	8-5
8.3.1.5	Peripheral Shut Down.....	8-6
8.3.2	Peripheral Behavior in Low-Power Modes .....	8-6
8.3.2.1	ColdFire Core .....	8-6
8.3.2.2	Static Random-Access Memory (SRAM) .....	8-6
8.3.2.3	System Control Module (SCM).....	8-6
8.3.2.4	DDR SDRAM Controller (SDRAMC).....	8-6
8.3.2.5	Chip Select Module .....	8-7
8.3.2.6	DMA Controller (DMA0–DMA3) .....	8-7
8.3.2.7	UART Modules (UART0, UART1, and UART2) .....	8-7
8.3.2.8	I2C Module.....	8-7
8.3.2.9	Queued Serial Peripheral Interface (QSPI).....	8-8
8.3.2.10	DMA Timers (DTIM0–DTIM3).....	8-8
8.3.2.11	Interrupt Controllers (INTC0, INTC1).....	8-8
8.3.2.12	Fast Ethernet Controller (FEC).....	8-9
8.3.2.13	I/O Ports.....	8-9
8.3.2.14	Reset Controller .....	8-9
8.3.2.15	Chip Configuration Module.....	8-9
8.3.2.16	Clock Module .....	8-10
8.3.2.17	Edge Port .....	8-10
8.3.2.18	Watchdog Timer .....	8-10
8.3.2.19	Programmable Interrupt Timers (PIT0, PIT1, PIT2 and PIT3) .....	8-10
8.3.2.20	USB Module .....	8-10
8.3.2.21	PWM Module .....	8-11
8.3.2.22	BDM .....	8-11
8.3.2.23	JTAG.....	8-11
8.3.3	Summary of Peripheral State During Low-Power Modes .....	8-11

## Chapter 9 Chip Configuration Module (CCM)

9.1	Introduction.....	9-1
9.1.1	Block Diagram.....	9-1

# Contents

Paragraph Number	Title	Page Number
9.1.2	Features .....	9-1
9.1.3	Modes of Operation .....	9-2
9.2	External Signal Descriptions .....	9-2
9.2.1	RCON .....	9-2
9.2.2	CLKMOD[1:0] .....	9-2
9.2.3	D[25:24, 21:19, 16] (Reset Configuration Override) .....	9-2
9.3	Memory Map/Register Definition .....	9-3
9.3.1	Programming Model .....	9-3
9.3.2	Memory Map .....	9-3
9.3.3	Register Descriptions .....	9-4
9.3.3.1	Chip Configuration Register (CCR) .....	9-4
9.3.3.2	Reset Configuration Register (RCON) .....	9-5
9.3.3.3	Chip Identification Register (CIR) .....	9-7
9.4	Functional Description .....	9-7
9.4.1	Reset Configuration .....	9-7
9.4.2	Chip Mode Selection .....	9-9
9.4.3	Boot Device Selection .....	9-10
9.4.4	Output Pad Strength Configuration .....	9-10
9.4.5	Clock Mode Selection .....	9-10
9.4.6	Chip Select Configuration .....	9-10
9.5	Reset .....	9-11

## Chapter 10 Reset Controller Module

10.1	Introduction .....	10-1
10.1.1	Block Diagram .....	10-1
10.1.2	Features .....	10-1
10.2	External Signal Description .....	10-2
10.2.1	RESET .....	10-2
10.2.2	RSTOUT .....	10-2
10.3	Memory Map/Register Definition .....	10-2
10.3.1	Reset Control Register (RCR) .....	10-2
10.3.2	Reset Status Register (RSR) .....	10-3
10.4	Functional Description .....	10-4
10.4.1	Reset Sources .....	10-4
10.4.1.1	Power-On Reset .....	10-5
10.4.1.2	External Reset .....	10-5
10.4.1.3	Watchdog Timer Reset .....	10-5
10.4.1.4	Loss-of-Clock Reset .....	10-5
10.4.1.5	Loss-of-Lock Reset .....	10-6

# Contents

Paragraph Number	Title	Page Number
10.4.1.6	Software Reset .....	10-6
10.4.2	Reset Control Flow .....	10-6
10.4.2.1	Synchronous Reset Requests .....	10-8
10.4.2.2	Internal Reset Request .....	10-8
10.4.2.3	Power-On Reset .....	10-8
10.4.3	Concurrent Resets .....	10-8
10.4.3.1	Reset Flow .....	10-8
10.4.3.2	Reset Status Flags .....	10-9

## Chapter 11 System Control Module (SCM)

11.1	Introduction .....	11-1
11.1.1	Overview .....	11-1
11.1.2	Features .....	11-1
11.2	Memory Map/Register Definition .....	11-2
11.2.1	Register Descriptions .....	11-3
11.2.1.1	Internal Peripheral System Base Address Register (IPSBAR) .....	11-3
11.2.1.2	Memory Base Address Register (RAMBAR) .....	11-4
11.2.1.3	Core Reset Status Register (CRSR) .....	11-6
11.2.1.4	Core Watchdog Control Register (CWCN) .....	11-7
11.2.1.5	Core Watchdog Service Register (CWSR) .....	11-8
11.3	Internal Bus Arbitration .....	11-9
11.3.1	Overview .....	11-10
11.3.2	Arbitration Algorithms .....	11-10
11.3.2.1	Round-Robin Mode .....	11-10
11.3.2.2	Fixed Mode .....	11-11
11.3.3	Bus Master Park Register (MPARK) .....	11-11
11.4	System Access Control Unit (SACU) .....	11-13
11.4.1	Overview .....	11-13
11.4.2	Features .....	11-13
11.4.3	Memory Map/Register Definition .....	11-14
11.4.3.1	Master Privilege Register (MPR) .....	11-15
11.4.3.2	Peripheral Access Control Registers (PACR0–PACR8) .....	11-15
11.4.3.3	Grouped Peripheral Access Control Register (GPACR) .....	11-17

## Chapter 12 General Purpose I/O Module

12.1	Introduction .....	12-1
------	--------------------	------



# Contents

Paragraph Number	Title	Page Number
12.1.1	Overview .....	12-3
12.1.2	Features .....	12-3
12.2	External Signal Description .....	12-3
12.3	Memory Map/Register Definition .....	12-10
12.3.1	Register Descriptions .....	12-11
12.3.1.1	Port Output Data Registers (PODR <sub>x</sub> ) .....	12-11
12.3.1.2	Port Data Direction Registers (PDDR <sub>x</sub> ) .....	12-14
12.3.1.3	Port Pin Data/Set Data Registers (PPDSDR <sub>x</sub> ) .....	12-16
12.3.1.4	Port Clear Output Data Registers (PCLRR <sub>x</sub> ) .....	12-19
12.3.1.5	Pin Assignment Registers (PAR <sub>x</sub> ) .....	12-21
12.3.1.6	Timer Pin Assignment Registers (PAR_TIMERH & PAR_TIMERL) .....	12-30
12.3.1.7	USB Pin Assignment Register (PAR_USB) .....	12-32
12.3.1.8	FEC0 & FEC1 Pin Assignment Registers (PAR_FEC0 & PAR_FEC1) .....	12-33
12.4	Functional Description .....	12-34
12.4.1	Overview .....	12-34
12.4.2	Port Digital I/O Timing .....	12-35
12.5	Initialization/Application Information .....	12-35

## Chapter 13 Interrupt Controller Modules

13.1	Introduction .....	13-1
13.1.1	68K/ColdFire Interrupt Architecture Overview .....	13-1
13.1.2	Interrupt Controller Theory of Operation .....	13-2
13.1.2.1	Interrupt Recognition .....	13-3
13.1.2.2	Interrupt Prioritization .....	13-3
13.1.2.3	Interrupt Vector Determination .....	13-4
13.2	Memory Map/Register Definition .....	13-4
13.2.1	Register Descriptions .....	13-6
13.2.1.1	Interrupt Pending Registers (IPRH <sub>n</sub> , IPRL <sub>n</sub> ) .....	13-6
13.2.1.2	Interrupt Mask Register (IMRH <sub>n</sub> , IMRL <sub>n</sub> ) .....	13-7
13.2.1.3	Interrupt Force Registers (INTFRCH <sub>n</sub> , INTFRCL <sub>n</sub> ) .....	13-9
13.2.1.4	Interrupt Request Level Register (IRLR <sub>n</sub> ) .....	13-11
13.2.1.5	Interrupt Acknowledge Level and Priority Register (IACKLPR <sub>n</sub> ) .....	13-11
13.2.1.6	Interrupt Control Register (ICR <sub>nx</sub> , (x = 1, 2,..., 63)) .....	13-12
13.2.1.7	Software and Level <i>n</i> IACK Registers (SWIACKR, L1IACK–L7IACK) .....	13-15
13.3	Prioritization Between Interrupt Controllers .....	13-16
13.4	Low-Power Wakeup Operation .....	13-16

# Contents

Paragraph Number	Title	Page Number
<b>Chapter 14</b>		
<b>Edge Port Module (EPORT)</b>		
14.1	Introduction.....	14-1
14.2	Low-Power Mode Operation .....	14-1
14.3	Interrupt/General-Purpose I/O Pin Descriptions.....	14-2
14.4	Memory Map/Register Definition .....	14-2
14.4.1	Register Description .....	14-3
14.4.1.1	EPORT Pin Assignment Register (EPPAR).....	14-3
14.4.1.2	EPORT Data Direction Register (EPDDR).....	14-4
14.4.1.3	Edge Port Interrupt Enable Register (EPIER) .....	14-5
14.4.1.4	Edge Port Data Register (EPDR).....	14-5
14.4.1.5	Edge Port Pin Data Register (EPPDR).....	14-6
14.4.1.6	Edge Port Flag Register (EPFR).....	14-6
<b>Chapter 15</b>		
<b>Chip Select Module</b>		
15.1	Introduction.....	15-1
15.1.1	Overview.....	15-1
15.2	External Signal Description .....	15-1
15.2.1	Chip Selects ( $\overline{CS}[7:0]$ ).....	15-1
15.2.2	Output Enable ( $\overline{OE}$ ) .....	15-1
15.2.3	Byte Strobes ( $\overline{BS}[3:2]$ ).....	15-2
15.3	Chip Select Operation.....	15-3
15.3.1	General Chip Select Operation .....	15-3
15.3.1.1	8- and 16-Bit Port Sizing .....	15-4
15.3.2	Enhanced Wait State Operation.....	15-4
15.3.2.1	External Boot Chip Select Operation .....	15-5
15.4	Memory Map/Register Definition .....	15-6
15.4.1	Chip Select Module Registers.....	15-7
15.4.1.1	Chip Select Address Registers (CSAR0–CSAR7) .....	15-7
15.4.1.2	Chip Select Mask Registers (CSMR0–CSMR7) .....	15-8
15.4.1.3	Chip Select Control Registers (CSCR0–CSCR7).....	15-9
15.5	Code Example.....	15-11
<b>Chapter 16</b>		
<b>External Interface Module (EIM)</b>		
16.1	Introduction.....	16-1

# Contents

Paragraph Number	Title	Page Number
16.1.1	Features .....	16-1
16.2	Bus and Control Signals .....	16-1
16.3	Bus Characteristics .....	16-2
16.4	Bus Errors .....	16-3
16.5	Data Transfer Operation .....	16-3
16.5.1	Bus Cycle Execution.....	16-4
16.5.2	Data Transfer Cycle States .....	16-5
16.5.3	Read Cycle.....	16-7
16.5.4	Write Cycle.....	16-8
16.5.5	Fast Termination Cycles .....	16-9
16.5.6	Back-to-Back Bus Cycles .....	16-10
16.5.7	Burst Cycles.....	16-11
16.5.7.1	Line Transfers .....	16-12
16.5.7.2	Line Read Bus Cycles.....	16-12
16.5.7.3	Line Write Bus Cycles.....	16-14
16.6	Secondary Wait State Operation.....	16-17
16.7	Misaligned Operands .....	16-17

## Chapter 17 SDRAM Controller (SDRAMC)

17.1	Introduction.....	17-1
17.1.1	Features .....	17-1
17.1.2	Terminology.....	17-1
17.1.3	Block Diagram.....	17-2
17.2	External Signal Description .....	17-4
17.3	Interface Recommendations .....	17-4
17.3.1	Supported Memory Configurations .....	17-4
17.3.2	SDRAM Connection Block Diagram .....	17-7
17.3.3	DDR SDRAM Layout Considerations .....	17-7
17.3.3.1	Termination Example .....	17-8
17.4	SDRAM Overview .....	17-8
17.4.1	SDRAM Commands.....	17-8
17.4.1.1	Activate Command (ACTV).....	17-9
17.4.1.2	Read Command (READ).....	17-9
17.4.1.3	Write Command (WRITE) .....	17-10
17.4.1.4	Precharge All Banks Command (PALL).....	17-10
17.4.1.5	Load Mode/Extended Mode Register Command (LMR, LEMR).....	17-10
17.4.1.6	Auto Refresh Command (REF) .....	17-12
17.4.1.7	Self Refresh (SREF) and Power Down (PDWN) Commands.....	17-12
17.4.2	Power-Up Initialization.....	17-13

# Contents

Paragraph Number	Title	Page Number
17.5	Memory Map/Register Definition .....	17-14
17.5.1	SDRAM Mode/Extended Mode Register (SDMR) .....	17-14
17.5.2	SDRAM Control Register (SDCR) .....	17-15
17.5.3	SDRAM Configuration Register 1 (SDCFG1) .....	17-17
17.5.4	SDRAM Configuration Register 2 (SDCFG2) .....	17-19
17.5.5	SDRAM Base Address Registers (SDBAR0 & SDBAR1) .....	17-20
17.5.6	SDRAM Address Mask Registers (SDMR0 & SDMR1) .....	17-20
17.6	Functional Overview .....	17-21
17.6.1	Page Management .....	17-21
17.6.2	Transfer Size .....	17-22
17.7	DDR SDRAM Example .....	17-23
17.7.1	SDRAM Chip Select Settings .....	17-24
17.7.2	SDRAM Configuration 1 Register Settings .....	17-25
17.7.3	SDRAM Configuration 2 Register Settings .....	17-26
17.7.4	SDRAM Control Register Settings and PALL command .....	17-27
17.7.5	Set the Extended Mode Register .....	17-28
17.7.6	Set the Mode Register and Reset DLL .....	17-29
17.7.7	Issue a PALL command .....	17-29
17.8	Perform Two Refresh Cycles .....	17-30
17.8.1	Clear the Reset DLL Bit in the Mode Register .....	17-32
17.8.2	Enable Automatic Refresh and Lock Mode Register .....	17-32
17.8.3	Initialization Code .....	17-33

## Chapter 18 DMA Controller Module

18.1	Introduction .....	18-1
18.1.1	Overview .....	18-1
18.1.2	Features .....	18-3
18.2	DMA Transfer Overview .....	18-4
18.3	Memory Map/Register Definition .....	18-5
18.3.1	DMA Request Control (DMAREQC) .....	18-6
18.3.2	Source Address Registers (SAR0–SAR3) .....	18-7
18.3.3	Destination Address Registers (DAR0–DAR3) .....	18-8
18.3.4	Byte Count Registers (BCR0–BCR3) and DMA Status Registers (DSR0–DSR3) .....	18-8
18.3.4.1	DMA Status Registers (DSR0–DSR3) .....	18-9
18.3.5	DMA Control Registers (DCR0–DCR3) .....	18-10
18.4	Functional Description .....	18-13
18.4.1	Transfer Requests (Cycle-Steal and Continuous Modes) .....	18-14
18.4.2	Dual-Address Data Transfer Mode .....	18-14
18.4.3	Channel Initialization and Startup .....	18-15

# Contents

Paragraph Number	Title	Page Number
18.4.3.1	Channel Prioritization .....	18-15
18.4.3.2	Programming the DMA Controller Module .....	18-15
18.4.4	Data Transfer .....	18-16
18.4.4.1	External Request and Acknowledge Operation .....	18-16
18.4.4.2	Auto-Alignment .....	18-19
18.4.4.3	Bandwidth Control .....	18-20
18.4.5	Termination .....	18-20

## Chapter 19 Fast Ethernet Controllers (FEC0 & FEC1)

19.1	Introduction .....	19-1
19.1.1	Overview .....	19-1
19.1.2	Block Diagram .....	19-1
19.1.3	Features .....	19-3
19.1.4	Modes of Operation .....	19-4
19.1.4.1	Full and Half Duplex Operation .....	19-4
19.1.5	Interface Options .....	19-4
19.1.5.1	10 Mbps and 100 Mbps MII Interface .....	19-4
19.1.5.2	10 Mbps 7-Wire Interface Operation .....	19-5
19.1.6	Address Recognition Options .....	19-5
19.1.7	Internal Loopback .....	19-5
19.2	Memory Map/Register Definition .....	19-5
19.2.1	High-Level Module Memory Map .....	19-5
19.2.2	Register Memory Map .....	19-6
19.2.3	MIB Block Counters Memory Map .....	19-7
19.2.4	Register Description .....	19-9
19.2.4.1	Ethernet Interrupt Event Register (EIR) .....	19-9
19.2.4.2	Interrupt Mask Registers (EIMR0 & EIMR1) .....	19-10
19.2.4.3	Receive Descriptor Active Registers (RDAR0 & RDAR1) .....	19-11
19.2.4.4	Transmit Descriptor Active Registers (TDAR0 & TDAR1) .....	19-12
19.2.4.5	Ethernet Control Registers (ECR0 & ECR1) .....	19-13
19.2.4.6	II Management Frame Registers (MMFR0 & MMFR1) .....	19-14
19.2.4.7	II Speed Control Registers (MSCR0 & MSCR1) .....	19-16
19.2.4.8	MIB Control Registers (MIBC0 & MIBC1) .....	19-17
19.2.4.9	Receive Control Registers (RCR0 & RCR1) .....	19-18
19.2.4.10	Transmit Control Registers (TCR0 & TCR1) .....	19-19
19.2.4.11	Physical Address Low Registers (PALR0 & PALR1) .....	19-20
19.2.4.12	Physical Address High Registers (PAUR0 & PAUR1) .....	19-21
19.2.4.13	Opcode/Pause Duration Registers (OPD0 & OPD1) .....	19-21
19.2.4.14	Descriptor Individual Upper Address Registers (IAUR0 & IAUR1) .....	19-22

# Contents

Paragraph Number	Title	Page Number
19.2.4.15	Descriptor Individual Lower Address Registers (IALR0 & IALR1) .....	19-23
19.2.4.16	Descriptor Group Upper Address Registers (GAUR0 & GAUR1).....	19-23
19.2.4.17	Descriptor Group Lower Address Registers (GALR0 & GALR1) .....	19-24
19.2.4.18	FIFO Transmit FIFO Watermark Registers (TFWR0 & TFWR1).....	19-25
19.2.4.19	FIFO Receive Bound Registers (FRBR0 & FRBR1) .....	19-25
19.2.4.20	FIFO Receive Start Registers (FRSR0 & FRSR1) .....	19-26
19.2.4.21	Receive Descriptor Ring Start Registers (ERDSR0 & ERDSR1).....	19-27
19.2.4.22	Transmit Buffer Descriptor Ring Start Registers (ETSDR0 & ETSDR1) .....	19-27
19.2.4.23	Receive Buffer Size Registers (EMRBR0 & EMRBR1) .....	19-28
19.2.5	Buffer Descriptors.....	19-29
19.2.5.1	Driver/DMA Operation with Buffer Descriptors .....	19-29
19.2.5.2	Ethernet Receive Buffer Descriptors (RxBD0 & RxBD1).....	19-31
19.2.5.3	Ethernet Transmit Buffer Descriptors (TxBD0 & TxBD1).....	19-33
19.3	Functional Description.....	19-35
19.3.1	Initialization Sequence.....	19-35
19.3.1.1	Hardware Controlled Initialization .....	19-35
19.3.2	User Initialization (Prior to Setting ECRn[ETHER_EN]).....	19-36
19.3.3	Microcontroller Initialization.....	19-36
19.3.4	User Initialization (After Asserting ECRn[ETHER_EN]) .....	19-37
19.3.5	Network Interface Options.....	19-37
19.3.6	FEC Frame Transmission .....	19-38
19.3.7	FEC Frame Reception.....	19-39
19.3.8	Ethernet Address Recognition .....	19-40
19.3.9	Hash Algorithm.....	19-43
19.3.10	Full Duplex Flow Control.....	19-46
19.3.11	Inter-Packet Gap (IPG) Time.....	19-47
19.3.12	Collision Handling .....	19-47
19.3.13	Internal and External Loopback.....	19-47
19.3.14	Ethernet Error-Handling Procedure .....	19-48
19.3.14.1	Transmission Errors .....	19-48
19.3.14.2	Reception Errors .....	19-49

## Chapter 20 Universal Serial Bus

20.1	Introduction.....	20-1
20.1.1	Block Diagram .....	20-1
20.1.2	Overview.....	20-1
20.1.3	Features.....	20-2
20.1.4	Modes of Operation .....	20-2
20.2	External Signal Description .....	20-3

# Contents

Paragraph Number	Title	Page Number
20.2.1	USB Clock (USB_CLK).....	20-3
20.2.2	USB Speed (USB_SPEED) .....	20-3
20.2.3	USB Negative/Positive Receive Signal Inputs (USB_RN & USB_RP) .....	20-4
20.2.4	USB Receive Data (USB_RXD) .....	20-4
20.2.5	USB Suspended (USB_SUSP) .....	20-4
20.2.6	USB Negative/Positive Transmit Signal Output (USB_TN & USB_TP) .....	20-4
20.2.7	USB Transmit Enable (USB_TXEN).....	20-4
20.3	Memory Map/Register Definition .....	20-4
20.3.1	Register Descriptions.....	20-5
20.3.1.1	USB Frame Number and Match Register (USB_FRAME).....	20-6
20.3.1.2	USB Specification/Release Number Register (USB_SPEC) .....	20-6
20.3.1.3	USB Status Register (USB_SR) .....	20-7
20.3.1.4	USB Control Register (USB_CR) .....	20-7
20.3.1.5	USB Descriptor RAM Address Register (USB_DAR) .....	20-9
20.3.1.6	USB Descriptor RAM/Endpoint Buffer Data Register (USB_DDR).....	20-10
20.3.1.7	USB Interrupt Status Register (USB_ISR).....	20-10
20.3.1.8	USB Interrupt Mask Register (USB_IMR) .....	20-12
20.3.1.9	USB FIFO Memory Control Register (USB_MCR) .....	20-13
20.3.1.10	Endpoint <i>n</i> Status/Control Register (USB_EPnSR).....	20-13
20.3.1.11	Endpoint <i>n</i> Interrupt Status Register (USB_EPnISR).....	20-15
20.3.1.12	Endpoint <i>n</i> Interrupt Mask Register (USB_EPnIMR) .....	20-17
20.3.1.13	Endpoint <i>n</i> FIFO Data Register (USB_EPnFDR).....	20-17
20.3.1.14	Endpoint <i>n</i> FIFO Status Register (USB_EPnFSR) .....	20-18
20.3.1.15	Endpoint <i>n</i> FIFO Control Register (USB_EPnFCR) .....	20-19
20.3.1.16	Endpoint <i>n</i> Last Read Frame Pointer (USB_EPnLRFP).....	20-21
20.3.1.17	Endpoint <i>n</i> Last Write Frame Pointer (USB_EPnLWFP).....	20-21
20.3.1.18	Endpoint <i>n</i> FIFO Alarm Register (USB_EPnFAR) .....	20-22
20.3.1.19	Endpoint <i>n</i> FIFO Read Pointer (USB_EPnFRP).....	20-23
20.3.1.20	Endpoint <i>n</i> FIFO Write Pointer (USB_EPnFWP).....	20-24
20.4	Functional Description.....	20-25
20.4.1	USB Components .....	20-25
20.4.1.1	Descriptor RAM .....	20-25
20.4.1.2	FIFO Controller .....	20-25
20.5	Reset Strategy .....	20-25
20.5.1	Description of Reset Operation.....	20-26
20.5.2	Hard Reset.....	20-26
20.5.2.1	USB Device Reset .....	20-26
20.5.2.2	USB Bus Reset .....	20-26
20.6	Interrupts.....	20-27
20.6.1	Description of Interrupt Operation .....	20-27
20.6.2	Detailed Interrupt Descriptions.....	20-27

# Contents

Paragraph Number	Title	Page Number
20.6.2.1	USB Global Interrupt.....	20-27
20.6.2.2	Endpoint Interrupts .....	20-29
20.6.2.3	Interrupts, Missed Interrupts, and the USB Device .....	20-30
20.7	Software Interface .....	20-31
20.7.1	Device Initialization.....	20-31
20.7.1.1	Configuration Download .....	20-32
20.7.1.2	USB Endpoint to FIFO Mapping.....	20-34
20.7.1.3	USB Descriptor Download .....	20-35
20.7.1.4	USB Interrupt Register .....	20-36
20.7.1.5	Endpoint Registers.....	20-36
20.7.1.6	FIFO Sizes .....	20-36
20.7.1.7	Enable the Device .....	20-36
20.7.2	Exception Handling .....	20-37
20.7.2.1	Unable to Complete Device Request.....	20-37
20.7.2.2	Aborted Device Request .....	20-37
20.7.2.3	Unable to Fill or Empty FIFO Due to Temporary Problem .....	20-37
20.7.2.4	Catastrophic Error.....	20-38
20.7.3	Data Transfer Operations.....	20-38
20.7.3.1	USB Packets .....	20-38
20.7.3.2	USB Transfers .....	20-39
20.7.3.3	Control Transfers .....	20-40
20.7.3.4	Bulk Traffic .....	20-41
20.7.3.5	Interrupt Traffic .....	20-42
20.7.3.6	Isochronous Operations .....	20-42

## Chapter 21 Watchdog Timer Module

21.1	Introduction.....	21-1
21.1.1	Low-Power Mode Operation .....	21-1
21.1.2	Block Diagram.....	21-2
21.2	Memory Map/Register Definition .....	21-2
21.2.1	Register Description .....	21-2
21.2.1.1	Watchdog Control Register (WCR).....	21-3
21.2.1.2	Watchdog Modulus Register (WMR).....	21-4
21.2.1.3	Watchdog Count Register (WCNTR).....	21-4
21.2.1.4	Watchdog Service Register (WSR) .....	21-4



# Contents

Paragraph Number	Title	Page Number
<b>Chapter 22</b>		
<b>Pulse Width Modulation (PWM) Module</b>		
22.1	Introduction.....	22-1
22.1.1	Overview.....	22-1
22.2	Memory Map/Register Definition .....	22-2
22.2.1	PWM Enable Register (PWME).....	22-2
22.2.2	PWM Polarity Register (PWMPOL) .....	22-3
22.2.3	PWM Clock Select Register (PWMCLK).....	22-4
22.2.4	PWM Prescale Clock Select Register (PWMPRCLK).....	22-4
22.2.5	PWM Center Align Enable Register (PWMCAE) .....	22-5
22.2.6	PWM Control Register (PWMCTL).....	22-6
22.2.7	PWM Scale A Register (PWMSCLA).....	22-7
22.2.8	PWM Scale B Register (PWMSCLB).....	22-7
22.2.9	PWM Channel Counter Registers (PWMCNTn).....	22-8
22.2.10	PWM Channel Period Registers (PWMPERn).....	22-9
22.2.11	PWM Channel Duty Registers (PWMDTYn) .....	22-10
22.3	Functional Description.....	22-10
22.3.1	PWM Clock Select.....	22-10
22.3.1.1	Prescaled Clock (A or B).....	22-11
22.3.1.2	Scaled Clock (SA or SB) .....	22-11
22.3.1.3	Clock Select .....	22-12
22.3.2	PWM Channel Timers .....	22-12
22.3.2.1	PWM Enable.....	22-13
22.3.2.2	PWM Polarity .....	22-13
22.3.2.3	PWM Period and Duty.....	22-13
22.3.2.4	PWM Timer Counters.....	22-14
22.3.2.5	Left Aligned Outputs .....	22-15
22.3.2.6	Center Aligned Outputs .....	22-16
22.3.2.7	PWM 16-Bit Functions.....	22-18
22.3.2.8	PWM Boundary Cases.....	22-19

## **Chapter 23**

### **Programmable Interrupt Timer Modules (PIT0–PIT3)**

23.1	Introduction.....	23-1
23.1.1	Overview.....	23-1
23.1.2	Block Diagram.....	23-1
23.1.3	Low-Power Mode Operation .....	23-2
23.2	Memory Map/Register Definition .....	23-2
23.2.1	Register Description .....	23-3

# Contents

Paragraph Number	Title	Page Number
23.2.1.1	PIT Control and Status Register (PCSR $n$ ).....	23-3
23.2.1.2	PIT Modulus Register (PMR $n$ ) .....	23-5
23.2.1.3	PIT Count Register (PCNTR $n$ ) .....	23-5
23.3	Functional Description.....	23-6
23.3.1	Set-and-Forget Timer Operation.....	23-6
23.3.2	Free-Running Timer Operation .....	23-6
23.3.3	Timeout Specifications .....	23-7
23.3.4	Interrupt Operation .....	23-7

## Chapter 24

### Queued Serial Peripheral Interface (QSPI) Module

24.1	Introduction.....	24-1
24.1.1	Overview.....	24-1
24.1.2	Features.....	24-1
24.1.3	Module Description .....	24-1
24.1.3.1	Interface and Signals.....	24-2
24.1.4	Internal Bus Interface.....	24-3
24.2	Operation .....	24-3
24.2.1	QSPI RAM.....	24-4
24.2.1.1	Receive RAM .....	24-5
24.2.1.2	Transmit RAM.....	24-5
24.2.1.3	Command RAM.....	24-6
24.2.2	Baud Rate Selection.....	24-6
24.2.3	Transfer Delays.....	24-7
24.2.4	Transfer Length.....	24-8
24.2.5	Data Transfer .....	24-8
24.3	Memory Map/Register Definition .....	24-9
24.3.1	QSPI Mode Register (QMR) .....	24-9
24.3.2	QSPI Delay Register (QDLYR) .....	24-11
24.3.3	QSPI Wrap Register (QWR).....	24-12
24.3.4	QSPI Interrupt Register (QIR).....	24-12
24.3.5	QSPI Address Register (QAR) .....	24-14
24.3.6	QSPI Data Register (QDR).....	24-14
24.3.7	Command RAM Registers (QCR0–QCR15).....	24-14
24.3.8	Programming Example .....	24-16

# Contents

Paragraph Number	Title	Page Number
<b>Chapter 25</b>		
<b>DMA Timers (DTIM0–DTIM3)</b>		
25.1	Introduction.....	25-1
25.1.1	Overview.....	25-1
25.1.2	Features.....	25-2
25.2	Memory Map/Register Definition .....	25-2
25.2.1	Prescaler.....	25-2
25.2.2	Capture Mode .....	25-3
25.2.3	Reference Compare.....	25-3
25.2.4	Output Mode .....	25-3
25.2.5	Memory Map .....	25-3
25.2.6	DMA Timer Mode Registers (DTMR <sub>n</sub> ).....	25-4
25.2.7	DMA Timer Extended Mode Registers (DTXMR <sub>n</sub> ).....	25-5
25.2.8	DMA Timer Event Registers (DTER <sub>n</sub> ).....	25-6
25.2.9	DMA Timer Reference Registers (DTRR <sub>n</sub> ).....	25-7
25.2.10	DMA Timer Capture Registers (DTCR <sub>n</sub> ) .....	25-8
25.2.11	DMA Timer Counters (DTCN <sub>n</sub> ) .....	25-8
25.3	Using the DMA Timer Modules.....	25-9
25.3.1	Code Example.....	25-10
25.3.2	Calculating Time-Out Values .....	25-11

## **Chapter 26** **I<sup>2</sup>C Interface**

26.1	Introduction.....	26-1
26.2	Overview.....	26-1
26.3	Features.....	26-1
26.4	I <sup>2</sup> C System Configuration.....	26-3
26.4.1	START Signal.....	26-3
26.4.2	Slave Address Transmission.....	26-4
26.4.3	Data Transfer .....	26-4
26.4.4	Acknowledge .....	26-4
26.4.5	STOP Signal .....	26-5
26.4.6	Repeated START .....	26-5
26.4.7	Clock Synchronization and Arbitration .....	26-6
26.4.8	Handshaking and Clock Stretching.....	26-8
26.5	Memory Map/Register Definition .....	26-8
26.5.1	I <sup>2</sup> C Address Register (I2ADR).....	26-8
26.5.2	I <sup>2</sup> C Frequency Divider Register (I2FDR).....	26-9
26.5.3	I <sup>2</sup> C Control Register (I2CR).....	26-10

# Contents

Paragraph Number	Title	Page Number
26.5.4	I <sup>2</sup> C Status Register (I2SR).....	26-11
26.5.5	I <sup>2</sup> C Data I/O Register (I2DR).....	26-12
26.6	I <sup>2</sup> C Programming Examples.....	26-13
26.6.1	Initialization Sequence.....	26-13
26.6.2	Generation of START.....	26-14
26.6.3	Post-Transfer Software Response.....	26-14
26.6.4	Generation of STOP.....	26-15
26.6.5	Generation of Repeated START.....	26-16
26.6.6	Slave Mode.....	26-16
26.6.7	Arbitration Lost.....	26-16

## Chapter 27 UART Modules

27.1	Introduction.....	27-1
27.1.1	Overview.....	27-1
27.1.2	Features.....	27-2
27.2	External Signal Description.....	27-3
27.3	Memory Map/Register Definition.....	27-4
27.3.1	UART Mode Registers 1 (UMR1 <i>n</i> ).....	27-5
27.3.2	UART Mode Register 2 (UMR2 <i>n</i> ).....	27-7
27.3.3	UART Status Registers (USR <i>n</i> ).....	27-8
27.3.4	UART Clock Select Registers (UCSR <i>n</i> ).....	27-10
27.3.5	UART Command Registers (UCR <i>n</i> ).....	27-10
27.3.6	UART Receive Buffers (URB <i>n</i> ).....	27-12
27.3.7	UART Transmit Buffers (UTB <i>n</i> ).....	27-12
27.3.8	UART Input Port Change Registers (UIPCR <i>n</i> ).....	27-13
27.3.9	UART Auxiliary Control Register (UACR <i>n</i> ).....	27-13
27.3.10	UART Interrupt Status/Mask Registers (UISR <i>n</i> /UIMR <i>n</i> ).....	27-14
27.3.11	UART Baud Rate Generator Registers (UBG1 <i>n</i> /UBG2 <i>n</i> ).....	27-15
27.3.12	UART Input Port Register (UIP <i>n</i> ).....	27-16
27.3.13	UART Output Port Command Registers (UOP1 <i>n</i> /UOP0 <i>n</i> ).....	27-16
27.4	Functional Description.....	27-17
27.4.1	Transmitter/Receiver Clock Source.....	27-17
27.4.1.1	Programmable Divider.....	27-17
27.4.1.2	Calculating Baud Rates.....	27-18
27.4.2	Transmitter and Receiver Operating Modes.....	27-19
27.4.2.1	Transmitter.....	27-19
27.4.2.2	Receiver.....	27-21
27.4.2.3	FIFO.....	27-23
27.4.3	Looping Modes.....	27-24

# Contents

Paragraph Number	Title	Page Number
27.4.3.1	Automatic Echo Mode .....	27-24
27.4.3.2	Local Loop-Back Mode .....	27-24
27.4.3.3	Remote Loop-Back Mode .....	27-25
27.4.4	Multidrop Mode .....	27-25
27.4.5	Bus Operation .....	27-27
27.4.5.1	Read Cycles .....	27-27
27.4.5.2	Write Cycles .....	27-27
27.4.6	Programming .....	27-27
27.4.6.1	Interrupt and DMA Request Initialization .....	27-28
27.4.6.2	UART Module Initialization Sequence .....	27-30

## Chapter 28 Message Digest Hardware Accelerator (MDHA)

28.1	Introduction .....	28-1
28.1.1	Overview .....	28-1
28.1.2	Features .....	28-1
28.1.3	Modes of Operation .....	28-2
28.2	Memory Map/Register Definition .....	28-3
28.2.1	MDHA Mode Register (MDMR) .....	28-4
28.2.1.1	Invalid Modes .....	28-6
28.2.2	MDHA Control Register (MDCR) .....	28-6
28.2.3	MDHA Command Register (MDCMR) .....	28-7
28.2.4	MDHA Status Register (MDSR) .....	28-8
28.2.5	MDHA Interrupt Status & Mask Registers (MDISR and MDIMR) .....	28-10
28.2.6	MDHA Data Size Register (MDDSR) .....	28-11
28.2.7	MDHA Input FIFO (MDIN) .....	28-12
28.2.8	MDHA Message Digest Registers 0 (MDx0) .....	28-12
28.2.9	MDHA Message Data Size Register (MDMDS) .....	28-12
28.2.10	MDHA Message Digest Registers 1 (MDx1) .....	28-13
28.3	Functional Description .....	28-13
28.3.1	MDHA Top Control .....	28-14
28.3.2	FIFO .....	28-14
28.3.3	MDHA Logic .....	28-14
28.3.3.1	Address Decoder .....	28-14
28.3.3.2	Interface Control .....	28-14
28.3.3.3	Auto-Padder .....	28-14
28.3.3.4	Hashing Engine .....	28-14
28.3.3.5	Hashing Engine Control .....	28-15
28.3.3.6	Status Interrupt .....	28-15
28.4	Initialization/Application Information .....	28-15

# Contents

Paragraph Number	Title	Page Number
28.4.1	Performing a Standard HASH Operation .....	28-15
28.4.2	Performing a HMAC Operation Without the MACFULL Bit .....	28-15
28.4.2.1	Generation of Key with IPAD .....	28-16
28.4.2.2	Generation of Key with OPAD.....	28-16
28.4.2.3	HMAC Hash .....	28-17
28.4.3	Performing a SHA-1 EHMAC.....	28-17
28.4.4	Performing a MAC Operation With the MACFULL Bit .....	28-18
28.4.5	Performing an NMAC .....	28-19

## Chapter 29 Symmetric Key Hardware Accelerator (SKHA)

29.1	Introduction.....	29-1
29.1.1	Features.....	29-1
29.1.2	Modes of Operation .....	29-1
29.1.2.1	Data Ecryption Standard (DES & 3DES) Algorithm .....	29-1
29.1.2.2	Advanced Encryption Standard (AES) Algorithm .....	29-3
29.1.2.3	Electronic Code Book (ECB) Cipher Mode .....	29-3
29.1.2.4	Cipher Block Chaining (CBC) Cipher Mode .....	29-4
29.1.2.5	Counter (CTR) Cipher Mode.....	29-5
29.2	Memory Map/Register Definition .....	29-6
29.2.1	Register Descriptions.....	29-7
29.2.1.1	SKHA Mode Register (SKMR).....	29-7
29.2.1.2	SKHA Control Register (SKCR).....	29-8
29.2.1.3	SKHA Command Register (SKCMR).....	29-9
29.2.1.4	SKHA Status Register (SKSR).....	29-10
29.2.1.5	SKHA Error Status Register (SKESR).....	29-11
29.2.1.6	SKHA Error Status Mask Register (SKESMR) .....	29-12
29.2.1.7	SKHA Key Size Register (SKKSR).....	29-13
29.2.1.8	SKHA Data Size Register (SKDSR) .....	29-13
29.2.1.9	SKHA Input FIFO .....	29-14
29.2.1.10	SKHA Output FIFO.....	29-14
29.2.1.11	SKHA Key Data Registers (SKKDR <sub>n</sub> ).....	29-14
29.2.1.12	SKHA Context Registers (SKCR <sub>n</sub> ).....	29-15
29.3	Functional Description.....	29-16
29.3.1	Transmit FIFO Interface Block.....	29-17
29.3.2	Receive FIFO Interface Block .....	29-17
29.3.3	Top Control Block .....	29-17
29.3.4	SKHA Logic Block.....	29-17
29.3.4.1	Address Decode Logic.....	29-18
29.3.4.2	Error Interrupt/Status Logic.....	29-18

# Contents

Paragraph Number	Title	Page Number
29.3.4.3	SKHA Core.....	29-18
29.3.5	Security Assurance Features.....	29-19
29.4	Initialization/Application Information.....	29-19
29.4.1	General Operation.....	29-19
29.4.2	Operation with Context Switch.....	29-20

## Chapter 30 Random Number Generator (RNG)

30.1	Introduction.....	30-1
30.1.1	Overview.....	30-1
30.2	Memory Map/Register Definition .....	30-1
30.2.1	RNG Control Register (RNGCR).....	30-1
30.2.2	RNG Status Register (RNGSR).....	30-2
30.2.3	RNG Entropy Register (RNGER).....	30-3
30.2.4	RNG Output FIFO (RNGOUT).....	30-4
30.3	Functional Description.....	30-5
30.3.1	Output FIFO.....	30-5
30.3.2	RNG Core/Control Logic Block.....	30-5
30.3.3	RNG Control Block .....	30-5
30.3.4	RNG Core Engine.....	30-6
30.4	Initialization/Application Information.....	30-6

## Chapter 31 Debug Support

31.1	Introduction.....	31-1
31.1.1	Overview.....	31-1
31.2	External Signal Description .....	31-2
31.3	Real-Time Trace Support.....	31-2
31.3.1	Begin Execution of Taken Branch (PST = 0x5).....	31-4
31.4	Memory Map/Register Definition .....	31-5
31.4.1	Revision A Shared Debug Resources .....	31-7
31.4.2	Address Attribute Trigger Register (AATR).....	31-7
31.4.3	Address Breakpoint Registers (ABLR, ABHR) .....	31-8
31.4.4	Configuration/Status Register (CSR).....	31-9
31.4.5	Data Breakpoint/Mask Registers (DBR, DBMR).....	31-12
31.4.6	Program Counter Breakpoint/Mask Registers (PBR, PBMR).....	31-13
31.4.7	Trigger Definition Register (TDR).....	31-14
31.5	Background Debug Mode (BDM) .....	31-16

# Contents

Paragraph Number	Title	Page Number
31.5.1	CPU Halt.....	31-16
31.5.2	BDM Serial Interface.....	31-17
31.5.2.1	Receive Packet Format .....	31-18
31.5.2.2	Transmit Packet Format.....	31-19
31.5.3	BDM Command Set.....	31-19
31.5.3.1	ColdFire BDM Command Format.....	31-21
31.5.3.2	Command Sequence Diagrams.....	31-22
31.5.3.3	Command Set Descriptions .....	31-23
31.6	Real-Time Debug Support .....	31-37
31.6.1	Theory of Operation.....	31-37
31.6.1.1	Emulator Mode .....	31-39
31.6.2	Concurrent BDM and Processor Operation .....	31-39
31.7	Processor Status, DDATA Definition.....	31-40
31.7.1	User Instruction Set .....	31-40
31.7.2	Supervisor Instruction Set.....	31-44
31.8	Freescall-Recommended BDM Pinout .....	31-45

## Chapter 32 IEEE 1149.1 Test Access Port (JTAG)

32.1	Introduction.....	32-1
32.1.1	Block Diagram .....	32-1
32.1.2	Features .....	32-2
32.1.3	Modes of Operation .....	32-3
32.2	External Signal Description .....	32-3
32.2.1	JTAG Enable (JTAG_EN).....	32-3
32.2.2	Test Clock Input (TCLK) .....	32-4
32.2.3	Test Mode Select/Breakpoint (TMS/BKPT).....	32-4
32.2.4	Test Data Input/Development Serial Input (TDI/DSI) .....	32-4
32.2.5	Test Reset/Development Serial Clock (TRST/DSCLK) .....	32-4
32.2.6	Test Data Output/Development Serial Output (TDO/DSO).....	32-5
32.3	Memory Map/Register Definition .....	32-5
32.3.1	Register Descriptions.....	32-5
32.3.1.1	Instruction Shift Register (IR) .....	32-5
32.3.1.2	IDCODE Register.....	32-5
32.3.1.3	Bypass Register .....	32-6
32.3.1.4	TEST_CTRL Register .....	32-6
32.3.1.5	Boundary Scan Register .....	32-7
32.4	Functional Description.....	32-7
32.4.1	JTAG Module .....	32-7
32.4.2	TAP Controller .....	32-7



# Contents

Paragraph Number	Title	Page Number
32.4.3	JTAG Instructions .....	32-8
32.4.3.1	EXTEST Instruction .....	32-9
32.4.3.2	IDCODE Instruction .....	32-9
32.4.3.3	SAMPLE/PRELOAD Instruction .....	32-9
32.4.3.4	ENABLE_TEST_CTRL Instruction .....	32-10
32.4.3.5	HIGHZ Instruction .....	32-10
32.4.3.6	CLAMP Instruction .....	32-10
32.4.3.7	BYPASS Instruction .....	32-10
32.5	Initialization/Application Information .....	32-11
32.5.1	Restrictions .....	32-11
32.5.2	Nonscan Chain Operation .....	32-11

## Appendix A Register Memory Map Quick Reference

# Contents

Paragraph Number	Title	Page Number
---------------------	-------	----------------

# About This Book

The primary objective of this reference manual is to define the functionality of the MCF5275 processor for use by software and hardware developers. In addition, this manual supports the MCF5274. This book is written from the perspective of the MCF5275, and unless otherwise noted, the information applies also to the MCF5274. The MCF5274 has the same functionality as the MCF5275 and any differences in data regarding bus timing, signal behavior, and AC, DC, and thermal characteristics are in the hardware specifications. Please refer to [Section 1.1, “MCF5275 Family Configurations,”](#) to see a summary of the differences.

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, it is the reader’s responsibility to be sure he is using 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/coldfire>.

## Audience

This manual is intended for system software and hardware developers and applications programmers who want to develop products with the MCF5275. It is assumed that the reader understands operating systems, microprocessor system design, basic principles of software and hardware, and basic details of the ColdFire® architecture.

## Organization

Following is a summary and brief description of the major sections of this manual:

- [Chapter 1, “Overview,”](#) includes general descriptions of the modules and features incorporated in the MCF5275, focussing in particular on new features.
- [Chapter 2, “Signal Descriptions,”](#) describes MCF5275 signals. It includes a listing of signals that characterizes each signal as an input or output, defines its state at reset, and identifies whether a pull-up resistor should be used.
- [Chapter 3, “ColdFire Core,”](#) provides an overview of the microprocessor core of the MCF5275. The chapter describes the organization of the Version 2 (V2) ColdFire processor core and an overview of the program-visible registers (the programming model) as they are implemented on the MCF5275.
- [Chapter 4, “Enhanced Multiply-Accumulate Unit \(EMAC\),”](#) describes the MCF5275 multiply/accumulate unit, which executes integer multiply, multiply-accumulate, and miscellaneous register instructions. The EMAC is integrated into the operand execution pipeline (OEP).

- [Chapter 5, “Cache,”](#) describes the MCF5275 cache implementation, including organization, configuration, and coherency. It describes cache operations and how the cache interacts with other memory structures.
- [Chapter 6, “Static RAM \(SRAM\),”](#) describes the MCF5275 on-chip static RAM (SRAM) implementation. It covers general operations, configuration, and initialization. It also provides information and examples of how to minimize power consumption when using the SRAM.
- [Chapter 7, “Clock Module,”](#) describes the MCF5275’s different clocking methods. It also describes clock module operation in low power modes.
- [Chapter 8, “Power Management,”](#) describes the low power operation of the MCF5275 and peripheral behavior in low power modes.
- [Chapter 9, “Chip Configuration Module \(CCM\),”](#) describes CCM functionality, detailing the two modes of chip operation: master mode and single-chip mode. This chapter provides a description of signals used by the CCM and a programming model.
- [Chapter 10, “Reset Controller Module,”](#) describes the operation of the reset controller module, detailing the different types of reset that can occur.
- [Chapter 11, “System Control Module \(SCM\),”](#) describes the functionality of the SCM, which provides the programming model for the System Access Control Unit (SACU), the system bus arbiter, a 32-bit Core Watchdog Timer (CWT), and the system control registers and logic.
- [Chapter 12, “General Purpose I/O Module,”](#) describes the operation and programming model of the general purpose I/O (GPIO) ports on the MCF5275.
- [Chapter 13, “Interrupt Controller Modules,”](#) describes operation of the interrupt controller portion of the SCM. Includes descriptions of the registers in the interrupt controller memory map and the interrupt priority scheme.
- [Chapter 14, “Edge Port Module \(EPORT\),”](#) describes EPORT module functionality, including operation in low power mode.
- [Chapter 15, “Chip Select Module,”](#) describes the MCF5275 chip-select implementation, including the operation and programming model, which includes the chip-select address, mask, and control registers.
- [Chapter 16, “External Interface Module \(EIM\),”](#) describes data-transfer operations, error conditions, bus arbitration, and reset operations.
- [Chapter 17, “SDRAM Controller \(SDRAMC\),”](#) describes the configuration and operation of the SDRAM controller. It begins with a general description and brief glossary, and includes a description of signals involved in DRAM operations. The remainder of the chapter describes the programming model and signal timing, as well as the command set required for synchronous operations.
- [Chapter 18, “DMA Controller Module,”](#) describes the MCF5275 Direct Memory Access (DMA) controller module. It provides an overview of the module and describes in detail its

signals and registers. The latter sections of this chapter describe operations, features, and supported data transfer modes in detail.

- [Chapter 19, “Fast Ethernet Controllers \(FEC0 & FEC1\),”](#) provides a feature-set overview, a functional block diagram, and transceiver connection information for both MII (Media Independent Interface) and 7-wire serial interfaces. It also provides describes operation and the programming model.
- [Chapter 20, “Universal Serial Bus,”](#) provides an overview of the universal serial bus (USB) device module. The *USB Specification, Revision 2.0* is a recommended supplement to this chapter.
- [Chapter 21, “Watchdog Timer Module,”](#) describes Watchdog timer functionality, including operation in low power mode.
- [Chapter 22, “Pulse Width Modulation \(PWM\) Module,”](#) describes the configuration and operation of the pulse width modulation (PWM) module. It includes a block diagram, programming model, and functional description.
- [Chapter 23, “Programmable Interrupt Timer Modules \(PIT0–PIT3\),”](#) describes the functionality of the four PIT timers, including operation in low power mode.
- [Chapter 25, “DMA Timers \(DTIM0–DTIM3\),”](#) describes the configuration and operation of the four DMA timer modules (DTIM0, DTIM1, DTIM2, and DTIM3). These 32-bit timers provide input capture and reference compare capabilities with optional signaling of events using interrupts or triggers. This chapter also provides programming examples.
- [Chapter 24, “Queued Serial Peripheral Interface \(QSPI\) Module,”](#) provides a feature-set overview and a description of operation, including details of the QSPI’s internal storage organization. The chapter concludes with the programming model and a timing diagram.
- [Chapter 27, “UART Modules,”](#) describes the use of the universal asynchronous receiver/transmitters (UARTs) implemented on the MCF5275 and includes programming examples.
- [Chapter 26, “I<sup>2</sup>C Interface,”](#) describes the MCF5275 I<sup>2</sup>C module, including I<sup>2</sup>C protocol, clock synchronization, and I<sup>2</sup>C programming model registers. It also provides extensive programming examples.
- [Chapter 28, “Message Digest Hardware Accelerator \(MDHA\),”](#) describes implementation of two of the world’s most popular cryptographic hash functions: SHA-1 and MD5. Accelerators for either algorithm separately have been designed, however the MDHA combines similar functions of the two algorithms into one small, optimized area of silicon on the MCF5275 device.
- [Chapter 30, “Random Number Generator \(RNG\),”](#) describes the 32-bit Random Number Generator (RNG), including a programming model, functional description, and application information.

- [Chapter 29, “Symmetric Key Hardware Accelerator \(SKHA\),”](#) describes the cryptographic hardware coprocessor designed to implement two widely used symmetric key block cipher algorithms, AES and DES.
- [Chapter 32, “IEEE 1149.1 Test Access Port \(JTAG\),”](#) describes configuration and operation of the MCF5275 Joint Test Action Group (JTAG) implementation. It describes those items required by the IEEE 1149.1 standard and provides additional information specific to the MCF5275. For internal details and sample applications, see the IEEE 1149.1 document.
- [Chapter 31, “Debug Support,”](#) describes the Revision A enhanced hardware debug support in the MCF5275.

This manual includes the following appendix:

- [Appendix A, “Register Memory Map Quick Reference,”](#) provides the entire address-map for MCF5275 memory-mapped registers.

## Suggested Reading

This section lists additional reading that provides background for the information in this manual as well as general information about the ColdFire architecture.

## Hardware Specification

The MCF5275EC document contains the mechanical and electrical specifications of the MCF5275. It can be found at <http://www.freescale.com/coldfire>.

## General Information

The following documentation provides useful information about the ColdFire architecture and computer architecture in general:

- *ColdFire Programmers Reference Manual, R1.0* (MCF5200PRM/AD)
- *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.

## ColdFire Documentation

ColdFire documentation is available from the sources listed on the back cover of this manual.

- Reference manuals (formerly called user's manuals)—These books provide details about individual ColdFire implementations and are intended to be used in conjunction with *The ColdFire 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.
- Hardware specifications—Hardware specifications provide specific data regarding 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 an implementation'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 ColdFire documentation, refer to <http://www.freescale.com/coldfire>.

## Conventions

This document uses the following notational conventions:

MNEMONICS	In text, instruction mnemonics are shown in uppercase.
mnemonics	In code and tables, instruction mnemonics are shown in lowercase.
<i>italics</i>	Italics indicate variable command parameters. Book titles in text are set in italics.
0x0	Prefix to denote hexadecimal number
0b0	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
word	A 16-bit data unit <sup>1</sup>
longword	A 32-bit data unit

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.

x	In some contexts, such as signal encodings, x indicates a don't care.
<i>n</i>	Used to express an undefined numerical value
~	NOT logical operator
&	AND logical operator
	OR logical operator

## 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 conversion
ALU	Arithmetic logic unit
BDM	Background debug mode
BIST	Built-in self test
BSDL	Boundary-scan description language
CODEC	Code/decode
DAC	Digital-to-analog conversion
DMA	Direct memory access
DSP	Digital signal processing
EA	Effective address
FIFO	First-in, first-out
GPIO	General-purpose I/O
I <sup>2</sup> C	Inter-integrated circuit
IEEE	Institute for Electrical and Electronics Engineers
IFP	Instruction fetch pipeline
IPL	Interrupt priority level
JEDEC	Joint Electron Device Engineering Council
JTAG	Joint Test Action Group
LIFO	Last-in, first-out
LRU	Least recently used
LSB	Least-significant byte
lsb	Least-significant bit
MAC	Multiply accumulate unit, also Media access controller
MBAR	Memory base address register



**Table i. Acronyms and Abbreviated Terms (Continued)**

Term	Meaning
MSB	Most-significant byte
msb	Most-significant bit
Mux	Multiplex
NOP	No operation
OEP	Operand execution pipeline
PC	Program counter
PCLK	Processor clock
PLIC	Physical layer interface controller
PLL	Phase-locked loop
POR	Power-on reset
PQFP	Plastic quad flat pack
PWM	Pulse width modulation
QSPI	Queued serial peripheral interface
RISC	Reduced instruction set computing
Rx	Receive
SIM	System integration module
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
<b>Opcode Wildcard</b>	
cc	Logical condition (example: NE for not equal)

**Table ii. Notational Conventions (Continued)**

Instruction	Operand Syntax
<b>Register Specifications</b>	
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)
<b>Register Names</b>	
ACC	MAC accumulator register
CCR	Condition code register (lower byte of SR)
MACSR	MAC status register
MASK	MAC mask register
PC	Program counter
SR	Status register
<b>Port Name</b>	
DDATA	Debug data port
PST	Processor status port
<b>Miscellaneous Operands</b>	
#<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	Both instruction and data caches
dc	Data cache
ic	Instruction cache

**Table ii. Notational Conventions (Continued)**

Instruction	Operand Syntax
# <vector>	Identifies the 4-bit vector number for trap instructions
<>	identifies an indirect data address referencing memory
<xxx>	identifies an absolute address referencing memory
dn	Signal displacement value, <i>n</i> 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
/	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.

**Table ii. Notational Conventions (Continued)**

Instruction	Operand Syntax
<b>Subfields and Qualifiers</b>	
{ }	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
<b>Condition Code Register Bit Names</b>	
C	Carry
N	Negative
V	Overflow
X	Extend
Z	Zero

## Revision History

Table iii provides a revision history for this document.

**Table iii. MCF5275RM Revision History**

Location	Substantive Changes
<b>Revision 0, 04/26/2004</b>	
—	Initial customer-release version.
<b>Revision 1.0, 08/16/2004</b>	
—	Various updates and formatting changes.
<b>Revision 1.1, 05/2005</b>	
—	Changes are noted in revision 1.3 or later of the MCF5275RMAD document.

**Table iii. MCF5275RM Revision History (Continued)**

Location	Substantive Changes
Revision 2, 07/2006	
—	Changes are noted in revision 1.8 or later of the MCF5275RMAD document.



# Chapter 1

## Overview

The MCF5275 family is a highly integrated implementation of the ColdFire® family of reduced instruction set computing (RISC) microprocessors. This document describes pertinent features and functions characteristics of the MCF5275 family. The MCF5275 family includes the MCF5275, MCF5275L, MCF5274 and MCF5274L microprocessors. The differences between these parts are summarized below in [Table 1-1](#). This document is written from the perspective of the MCF5275 and unless otherwise noted, the information applies also to the MCF5275L, MCF5274 and MCF5274L.

The MCF5275 family delivers a new level of performance and integration on the popular version 2 ColdFire core with up to 144 (Dhrystone 2.1) MIPS @ 150MHz. These highly integrated microprocessors build upon the widely used peripheral mix on the popular MCF5272 ColdFire microprocessor (10/100 Mbps Ethernet MAC and USB device) by adding a second 10/100 Mbps Ethernet MAC (MCF5274 & MCF5275) and hardware encryption (MCF5275L and MCF5275). In addition, the MCF5275 family features an Enhanced Multiply Accumulate Unit (EMAC), large on-chip memory (64 Kbytes SRAM, 16 Kbytes configurable cache), and a 16-bit DDR SDRAM memory controller.

These devices are ideal for cost-sensitive applications requiring significant control processing for file management, connectivity, data buffering, and user interface, as well as signal processing in a variety of key markets such as security, imaging, networking, gaming, and medical. This leading package of integration and high performance allows fast time to market through easy code reuse and extensive third party tool support.

To locate any published errata or updates for this document, refer to the ColdFire products website at <http://www.freescale.com/coldfire>.

## 1.1 MCF5275 Family Configurations

**Table 1-1. MCF5275 Family Configurations**

Module	5274L	5275L	5274	5275
ColdFire Version 2 Core with EMAC (Enhanced Multiply-Accumulate Unit)	x	x	x	x
System Clock	up to 150 MHz			
Performance (Dhrystone/2.1 MIPS)	144			
Instruction/Data Cache	16 Kbytes (configurable)			
Static RAM (SRAM)	64 Kbytes			
Interrupt Controllers (INTC)	2	2	2	2
Edge Port Module (EPORT)	x	x	x	x

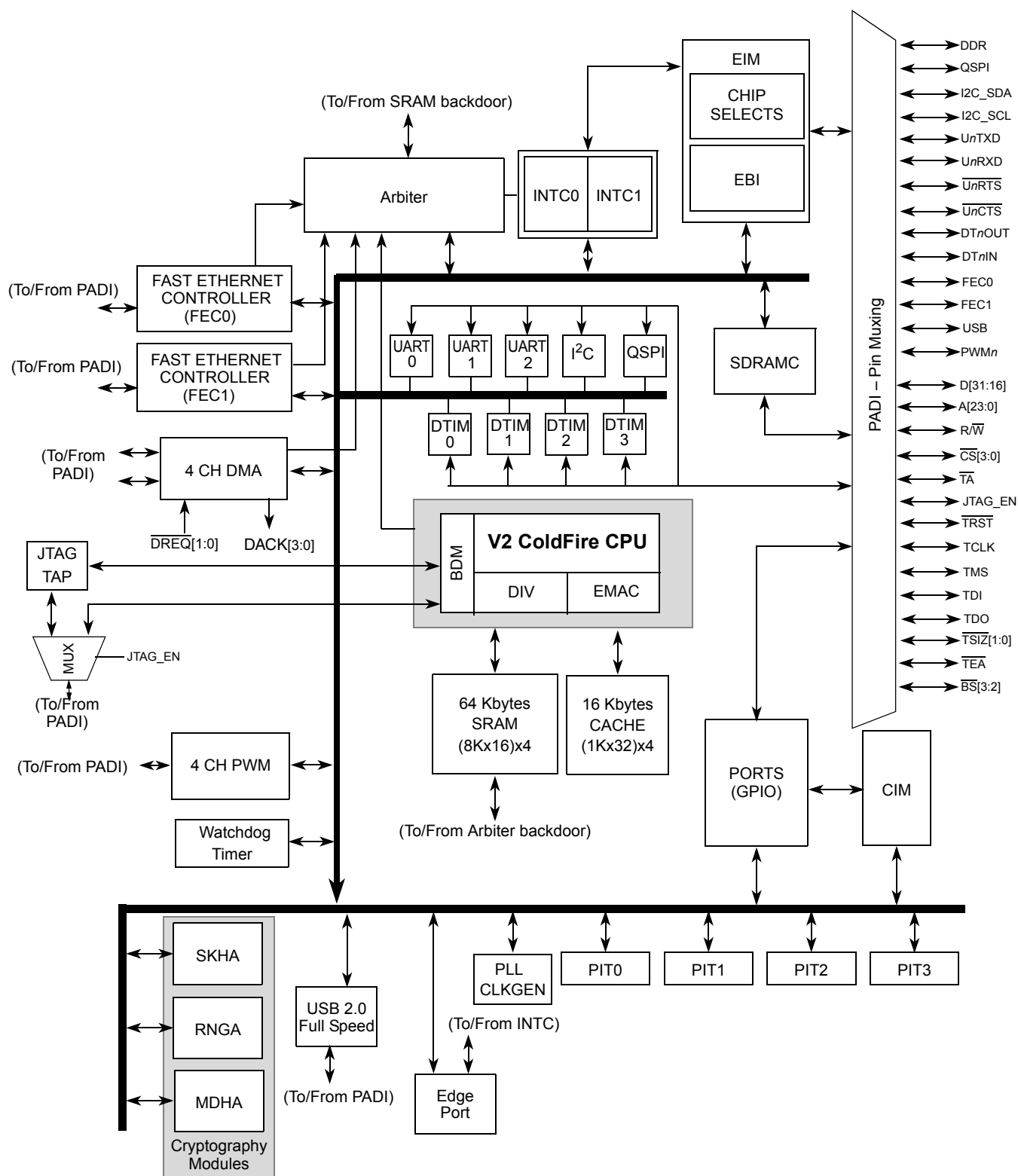
**Table 1-1. MCF5275 Family Configurations (Continued)**

Module	5274L	5275L	5274	5275
External Interface Module (EIM)	x	x	x	x
4-channel Direct-Memory Access (DMA)	x	x	x	x
DDR SDRAM Controller	x	x	x	x
Fast Ethernet Controller (FEC)	1	1	2	2
Watchdog Timer Module (WDT)	x	x	x	x
4-channel Programmable Interval Timer Module (PIT)	x	x	x	x
32-bit DMA Timers	4	4	4	4
USB	x	x	x	x
QSPI	x	x	x	x
UART(s)	up to 3	up to 3	3	3
I <sup>2</sup> C	x	x	x	x
PWM	4	4	4	4
General Purpose I/O Module (GPIO)	x	x	x	x
CIM = Chip Configuration Module + Reset Controller Module	x	x	x	x
Debug BDM	x	x	x	x
JTAG - IEEE 1149.1 Test Access Port	x	x	x	x
Cryptography Hardware Accelerators	—	x	—	x
Package	196 MAPBGA	196 MAPBGA	256 MAPBGA	256 MAPBGA

## 1.2 Block Diagram

The superset device in the MCF5275 family comes in a 256 mold array process ball grid array (MAPBGA) package. [Figure 1-1](#) shows a top-level block diagram of the MCF5275.





### Figure 1-1. MCF5275 Block Diagram

## 1.3 Features

### 1.3.1 Feature Overview

- Version 2 ColdFire variable-length RISC processor core
  - Static operation
  - 32-bit address and data path on-chip
  - Processor core runs at twice the internal bus frequency
  - Sixteen general-purpose 32-bit data and address registers
  - Implements the ColdFire Instruction Set Architecture, ISA\_A, with extensions to support the user stack pointer register, and 4 new instructions for improved bit processing
  - Enhanced Multiply-Accumulate (EMAC) unit with four 48-bit accumulators to support 32-bit signal processing algorithms
  - Illegal instruction decode that allows for 68K emulation support
- System debug support
  - Real time trace for determining dynamic execution path
  - Background debug mode (BDM) for in-circuit debugging
  - Real time debug support, with two user-visible hardware breakpoint registers (PC and address with optional data) that can be configured into a 1- or 2-level trigger
- On-chip memories
  - 16-Kbyte cache, configurable as instruction-only, data-only, or split I-/D-cache
  - 64-Kbyte dual-ported SRAM on CPU internal bus, accessible by core and non-core bus masters (e.g., DMA, FEC)
- Two Fast Ethernet Controllers (FEC)
  - 10 BaseT capability, half duplex or full duplex
  - 100 BaseT capability, half duplex or full duplex
  - On-chip transmit and receive FIFOs
  - Built-in dedicated DMA controller
  - Memory-based flexible descriptor rings
  - Media independent interface (MII) to external transceiver (PHY)
- USB Device Module
  - Supports full-speed 12-Mbps and low-speed 1.5-Mbps USB devices
  - Automatic hardware processing of USB standard device requests
  - Requires external transceiver

- Protocol control and administration for up to four endpoints (programmable types)
- One FIFO RAM per endpoint (2-Kbyte total)
- Dedicated 1-Kbyte descriptor RAM
- Remote wake-up
- Three Universal Asynchronous Receiver Transmitters (UARTs)
  - 16-bit divider for clock generation
  - Interrupt control logic
  - Maskable interrupts
  - DMA support
  - Data formats can be 5, 6, 7 or 8 bits with even, odd or no parity
  - Up to 2 stop bits in 1/16 increments
  - Error-detection capabilities
  - Modem support includes request-to-send ( $\overline{UnRTS}$ ) and clear-to-send ( $\overline{UnCTS}$ ) lines for two UARTs
  - Transmit and receive FIFO buffers
- I<sup>2</sup>C Module
  - Interchip bus interface for EEPROMs, LCD controllers, A/D converters, and keypads
  - Fully compatible with industry-standard I<sup>2</sup>C bus
  - Master or slave modes support multiple masters
  - Automatic interrupt generation with programmable level
- Queued Serial Peripheral Interface (QSPI)
  - Full-duplex, three-wire synchronous transfers
  - Up to four chip selects available
  - Master mode operation only
  - Programmable master bit rates
  - Up to 16 pre-programmed transfers
- Four 32-bit DMA Timers
  - 12-ns resolution at 83 MHz
  - Programmable sources for clock input, including an external clock option
  - Programmable prescaler
  - Input-capture capability with programmable trigger edge on input pin
  - Output-compare with programmable mode for the output pin
  - Free run and restart modes

- Maskable interrupts on input capture or reference-compare
- DMA trigger capability on input capture or reference-compare
- Pulse width modulation (PWM) unit
  - Four independent channels with programmable period and duty cycle
- Four Periodic Interrupt Timers (PITs)
  - 16-bit counter
  - Selectable as free running or count down
- Software Watchdog Timer
  - 16-bit counter
  - Low power mode support
- Phase Locked Loop (PLL)
  - Crystal or external oscillator reference
  - 8 to 25 MHz reference frequency for normal PLL mode
  - 24 to 75 MHz oscillator reference frequency for 1:1 mode (input freq = core freq =  $2 \times \text{CLKOUT}$ )
  - Separate clock output pins
- Interrupt Controllers (x2)
  - Support for up to 110 interrupt sources per interrupt controller organized as follows:
    - 103 fully-programmable interrupt sources
    - 7 fixed-level interrupt sources
  - Seven external interrupt signals
  - Unique vector number for each interrupt source
  - Ability to mask any individual interrupt source or all interrupt sources (global mask-all)
  - Support for hardware and software interrupt acknowledge (IACK) cycles
  - Combinatorial path to provide wake-up from low power modes
- DMA Controller
  - Four fully programmable channels
  - Dual-address transfer support with 8-, 16- and 32-bit data capability along with support for 16-byte (4 x 32-bit) burst transfers
  - Source/destination address pointers that can increment or remain constant
  - 24-bit byte transfer counter per channel
  - Auto-alignment transfers supported for efficient block movement
  - Bursting and cycle steal support

- Software-programmable connections between the four DMA channels and the 14 DMA requesters in the UARTs (6), 32-bit timers (4), and external logic (4)
- External Bus Interface
  - Glueless connections to external memory devices (e.g., SRAM, Flash, ROM, etc.)
  - Support for n-1-1-1 burst fetches from page mode Flash
  - Glueless interface to SRAM devices with or without byte strobe inputs
  - Programmable wait state generator
  - 16-bit bidirectional data bus
  - 24-bit address bus
  - Up to eight chip selects available
  - Byte/write enables (byte strobes)
  - Ability to boot from external memories that are 8 or 16 bits wide
- DDR SDRAM controller
  - Supports 16-bit wide memory devices
  - Supports Dual Data Rate (DDR) SDRAM.
  - Page mode support
  - Programmable refresh interval timer.
  - Sleep mode and self-refresh.
  - Supports 16-byte (4-beat, 4-byte or 8-beat, 2-byte) critical-word-first burst transfer.
  - Up to two chip selects available supporting 8 Mbyte to 128 MByte each.
  - 150 MHz data transfer rate (DDR)
- Chip Integration Module (CIM)
  - System configuration during reset
  - Selects one of four clock modes
  - Sets boot device and its data port width
  - Configures output pad drive strength
  - Unique part identification number and part revision number
  - Reset
    - Separate reset in and reset out signals
    - Six sources of reset: Power-on reset (POR), External, Software, Watchdog, PLL loss of clock, PLL loss of lock
    - Status flag indication of source of last reset
- General Purpose I/O interface
  - Up to 65 bits of general purpose I/O

- Bit manipulation supported via set/clear functions
- Unused peripheral pins may be used as extra GPIO
- JTAG support for system level board testing

### 1.3.2 V2 Core Overview

The processor core is comprised of two separate pipelines that are decoupled by an instruction buffer. The two-stage Instruction Fetch Pipeline (IFP) is responsible for instruction-address generation and instruction fetch. The instruction buffer is a first-in-first-out (FIFO) buffer that holds prefetched instructions awaiting execution in the Operand Execution Pipeline (OEP). The OEP includes two pipeline stages. The first stage decodes instructions and selects operands (DSOC); the second stage (AGEX) performs instruction execution and calculates operand effective addresses, if needed.

The V2 core implements the ColdFire Instruction Set Architecture Revision A with added support for a separate user stack pointer register and four new instructions to assist in bit processing. Additionally, the MCF5275 core includes the enhanced multiply-accumulate unit (EMAC) for improved signal processing capabilities. The EMAC implements a 4-stage execution pipeline, optimized for 32 x 32 bit operations, with support for four 48-bit accumulators. Supported operands include 16- and 32-bit signed and unsigned integers as well as signed fractional operands and a complete set of instructions to process these data types. The EMAC provides superb support for execution of DSP operations within the context of a single processor at a minimal hardware cost.

### 1.3.3 Integrated Debug Module

The ColdFire processor core debug interface is provided to support system debugging in conjunction with low-cost debug and emulator development tools. Through a standard debug interface, users can access real-time trace and debug information. This allows the processor and system to be debugged at full speed without the need for costly in-circuit emulators. The debug interface is a superset of the BDM interface provided on the 683xx family of parts.

The on-chip breakpoint resources include a total of 8 programmable registers—a set of address registers (with two 32-bit registers), a set of data registers (with a 32-bit data register plus a 32-bit data mask register), an address attribute register, a trigger definition register, and one 32-bit PC register plus a 32-bit PC mask register. These registers can be accessed through the dedicated debug serial communication channel or from the processor's supervisor mode programming model. The breakpoint registers can be configured to generate triggers by combining the address, data, and PC conditions in a variety of single or dual-level definitions. The trigger event can be programmed to generate a processor halt or initiate a debug interrupt exception.

To support program trace, the Version 2 debug module provides processor status (PST[3:0]) and debug data (DDATA[3:0]) ports. These buses and the PSTCLK output provide execution status,

captured operand data, and branch target addresses defining processor activity at the CPU's clock rate.

### 1.3.4 JTAG

The MCF5275 supports circuit board test strategies based on the Test Technology Committee of IEEE and the Joint Test Action Group (JTAG). The test logic includes a test access port (TAP) consisting of a 16-state controller, an instruction register, and three test registers (a 1-bit bypass register, a 326-bit boundary-scan register, and a 32-bit ID register). The boundary scan register links the device's pins into one shift register. Test logic, implemented using static logic design, is independent of the device system logic.

The MCF5275 implementation can do the following:

- Perform boundary-scan operations to test circuit board electrical continuity
- Sample MCF5275 system pins during operation and transparently shift out the result in the boundary scan register
- Bypass the MCF5275 for a given circuit board test by effectively reducing the boundary-scan register to a single bit
- Disable the output drive to pins during circuit-board testing
- Drive output pins to stable levels

### 1.3.5 On-chip Memories

#### 1.3.5.1 Cache

The 16-Kbyte cache can be configured into one of three possible organizations: an 16-Kbyte instruction cache, an 16-Kbyte data cache or a split 8-Kbyte instruction/8-Kbyte data cache. The configuration is software-programmable by control bits within the privileged Cache Configuration Register (CACR). In all configurations, the cache is a direct-mapped single-cycle memory, organized as 1024 lines, each containing 16 bytes of data. The memories consist of a 1024-entry tag array (containing addresses and control bits) and a 16-Kbyte data array, organized as 4096 x 32 bits.

If the desired address is mapped into the cache memory, the output of the data array is driven onto the ColdFire core's local data bus, completing the access in a single cycle. If the data is not mapped into the tag memory, a cache miss occurs and the processor core initiates a 16-byte line-sized fetch. The cache module includes a 16-byte line fill buffer used as temporary storage during miss processing. For all data cache configurations, the memory operates in write-through mode and all operand writes generate an external bus cycle.

### 1.3.5.2 SRAM

The SRAM module provides a general-purpose 64-Kbyte memory block that the ColdFire core can access in a single cycle. The location of the memory block can be set to any 64-Kbyte boundary within the 4-Gbyte address space. The memory is ideal for storing critical code or data structures, for use as the system stack, or for storing FEC data buffers. Because the SRAM module is physically connected to the processor's high-speed local bus, it can quickly service core-initiated accesses or memory-referencing commands from the debug module.

The SRAM module is also accessible by the DMA and FEC non-core bus masters. The dual-ported nature of the SRAM makes it ideal for implementing applications with double-buffer schemes, where the processor and a DMA device operate in alternate regions of the SRAM to maximize system performance. As an example, system performance can be increased significantly if Ethernet packets are moved from the FEC into the SRAM (rather than external memory) prior to any processing.

### 1.3.6 Fast Ethernet Controller (FEC)

The MCF5275 contains up to two Fast Ethernet Controllers (FEC) which perform the full set of IEEE 802.3/Ethernet CSMA/CD media access control and channel interface functions. The FECs support connection and functionality for the 10/100 Mbps 802.3 media independent interface (MII). It requires an external transceiver (PHY) to complete the interface to the media.

### 1.3.7 Universal Serial Bus (USB)

The USB controller supports device mode data communications with a USB host (typically a PC). The programmable USB registers allow the user to enable or disable the module, control characteristics of individual endpoints, and monitor traffic flow through the module without ever seeing the low-level details of the USB protocol.

### 1.3.8 UARTs

The MCF5275 contains three full-duplex UARTs that function independently. The three UARTs can be clocked by the system bus clock, eliminating the need for an externally supplied clock. They can use DMA requests on transmit-ready and receive-ready as well as interrupt requests for servicing. Flow control via  $\overline{UnCTS}$  and  $\overline{UnRTS}$  pins is provided on all three UARTS.

### 1.3.9 I<sup>2</sup>C Bus

The I<sup>2</sup>C bus is a two-wire, bidirectional serial bus that provides a simple, efficient method of data exchange, minimizing the interconnection between devices. This bus is suitable for applications requiring occasional communications over a short distance between many devices.



### 1.3.10 QSPI

The queued serial peripheral interface module provides a high-speed synchronous serial peripheral interface with queued transfer capability. It allows up to 16 transfers to be queued at once, eliminating CPU intervention between transfers.

### 1.3.11 Cryptography

The superset device, MCF5275, incorporates small, fast, dedicated hardware accelerators for random number generation, message digest and hashing, and the DES, 3DES, and AES block cipher functions allowing for the implementation of common Internet security protocol cryptography operations with performance well in excess of software-only algorithms.

### 1.3.12 DMA Timers (DTIM0-DTIM3)

There are four independent, DMA-transfer-generating 32-bit timers (DTIM[3:0]) on the MCF5275. Each timer module incorporates a 32-bit timer with a separate register set for configuration and control. The timers can be configured to operate from the system clock or from an external clock source using one of the DTIN $n$  signals. If the system clock is selected, it can be divided by 16 or 1. The input clock is further divided by a user-programmable 8-bit prescaler which clocks the actual timer counter register (TCR $n$ ). Each of these timers can be configured for input capture or reference compare mode. By configuring the internal registers, each timer may be configured to assert an external signal, generate an interrupt on a particular event or cause a DMA transfer.

### 1.3.13 Pulse Width Modulation (PWM) Module

The Pulse Width Modulation (PWM) module generates a synchronous series of pulses having programmable duty cycle. With a suitable low-pass filter, the PWM can be used as a digital-to-analog converter.

The PWM module has four channels, each having a programmable period and duty cycle as well as a dedicated counter. A flexible clock select scheme allows a total of four different clock sources to be used with the counters. Each of the modulators can create independent continuous waveforms with software-selectable duty rates from 0% to 100%. The PWM outputs can be programmed as left aligned outputs or center aligned outputs

### 1.3.14 Periodic Interrupt Timers (PIT0-PIT3)

The four periodic interrupt timers (PIT[3:0]) are 16-bit timers that provide precise interrupts at regular intervals with minimal processor intervention. Each timer can either count down from the value written in its PIT modulus register, or it can be a free-running down-counter.

### 1.3.15 Software Watchdog Timer

The watchdog timer is a 16-bit timer that facilitates recovery from runaway code. The watchdog counter is a free-running down-counter that generates a reset on underflow. To prevent a reset, software must periodically restart the countdown.

### 1.3.16 Clock Module and Phase Locked Loop (PLL)

The clock module contains a crystal oscillator (OSC), phase-locked loop (PLL), reduced frequency divider (RFD), status/control registers, and control logic. To improve noise immunity, the PLL and OSC have their own power supply inputs, VDDPLL and VSSPLL. All other circuits are powered by the normal supply pins, VDD, VSS, OVDD, and OVSS.

### 1.3.17 Interrupt Controllers (INTC0, INTC1)

There are two interrupt controllers on the MCF5275, each of which can support up to 63 interrupt sources each for a total of 126. Each interrupt controller is organized as 7 levels with 9 interrupt sources per level. Each interrupt source has a unique interrupt vector, and 56 of the 63 sources of a given controller provide a programmable level [1-7] and priority within the level.

### 1.3.18 DMA Controller

The Direct Memory Access (DMA) Controller Module provides an efficient way to move blocks of data with minimal processor interaction. The DMA module provides four channels (DMA0-DMA3) that allow byte, word, longword or 16-byte burst line transfers. These transfers are triggered by software explicitly setting a DCR<sub>n</sub>[START] bit. Other sources include the DMA timer, external sources via the  $\overline{\text{DREQ}}$  signal, and UARTs. The DMA controller supports dual address to off-chip or on-chip devices.

### 1.3.19 External Interface Module (EIM)

The external bus interface handles the transfer of information between the core and memory, peripherals, or other processing elements in the external address space. Features have been added to support external Flash modules, for secondary wait states on reads and writes, and a signal to support Active-Low Address Valid (a signal on most Flash memories).

Programmable chip-select outputs provide signals to enable external memory and peripheral circuits, providing all handshaking and timing signals for automatic wait-state insertion and data bus sizing.

Base memory address and block size are programmable, with some restrictions. For example, the starting address must be on a boundary that is a multiple of the block size. Each chip select can be configured to provide read and write enable signals suitable for use with most popular static RAMs

and peripherals. Data bus width (8-bit or 16-bit) is programmable on all chip selects, and further decoding is available for protection from read-only access.

### 1.3.20 Double Data Rate (DDR) Synchronous DRAM (SDRAM) Controller

The SDRAMC provides a 16-bit external interface to double-data-rate (DDR) SDRAM memory devices. It enables a glueless interface to DDR SDRAM memory devices. It is responsible for providing address, data and control signals for up to two independent chip-selects.

### 1.3.21 Reset

The reset controller is provided to determine the cause of reset, assert the appropriate reset signals to the system, and keep track of what caused the last reset. The power management registers for the internal low-voltage detect (LVD) circuit are implemented in the reset module. There are six sources of reset:

- External
- Power-on reset (POR)
- Watchdog timer
- Phase locked-loop (PLL) loss of lock
- PLL loss of clock
- Software

External reset on the  $\overline{\text{RSTOUT}}$  pin is software-assertable independent of chip reset state. There are also software-readable status flags indicating the cause of the last reset.

### 1.3.22 GPIO

Like the MC68332, unused bus interface and peripheral pins on the MCF5275 can be used as discrete general-purpose inputs and outputs. These are managed by a dedicated GPIO module that logically groups all pins into ports located within a contiguous block of memory-mapped control registers.

All of the pins associated with the external bus interface may be used for several different functions. Their primary function is to provide an external memory interface to access off-chip resources. When not used for this, all of the pins may be used as general-purpose digital I/O pins. In some cases, the pin function is set by the operating mode, and the alternate pin functions are not supported.

The digital I/O pins on the MCF5275 are grouped into 8-bit ports. Some ports do not use all eight bits. Each port has registers that configure, monitor, and control the port pins.

## 1.4 Documentation

Documentation is available from a local Freescale distributor, a Freescale sales office, the Freescale Literature Distribution Center, or through the Freescale world-wide web address at <http://www.freescale.com/coldfire>.

# Chapter 2

## Signal Descriptions

### 2.1 Introduction

This chapter describes MCF5275 signals. It includes an alphabetical listing of signals that characterizes each signal as an input or output, defines its state at reset, and identifies whether a pull-up resistor should be used. [Chapter 16, “External Interface Module \(EIM\),”](#) describes how these signals interact.

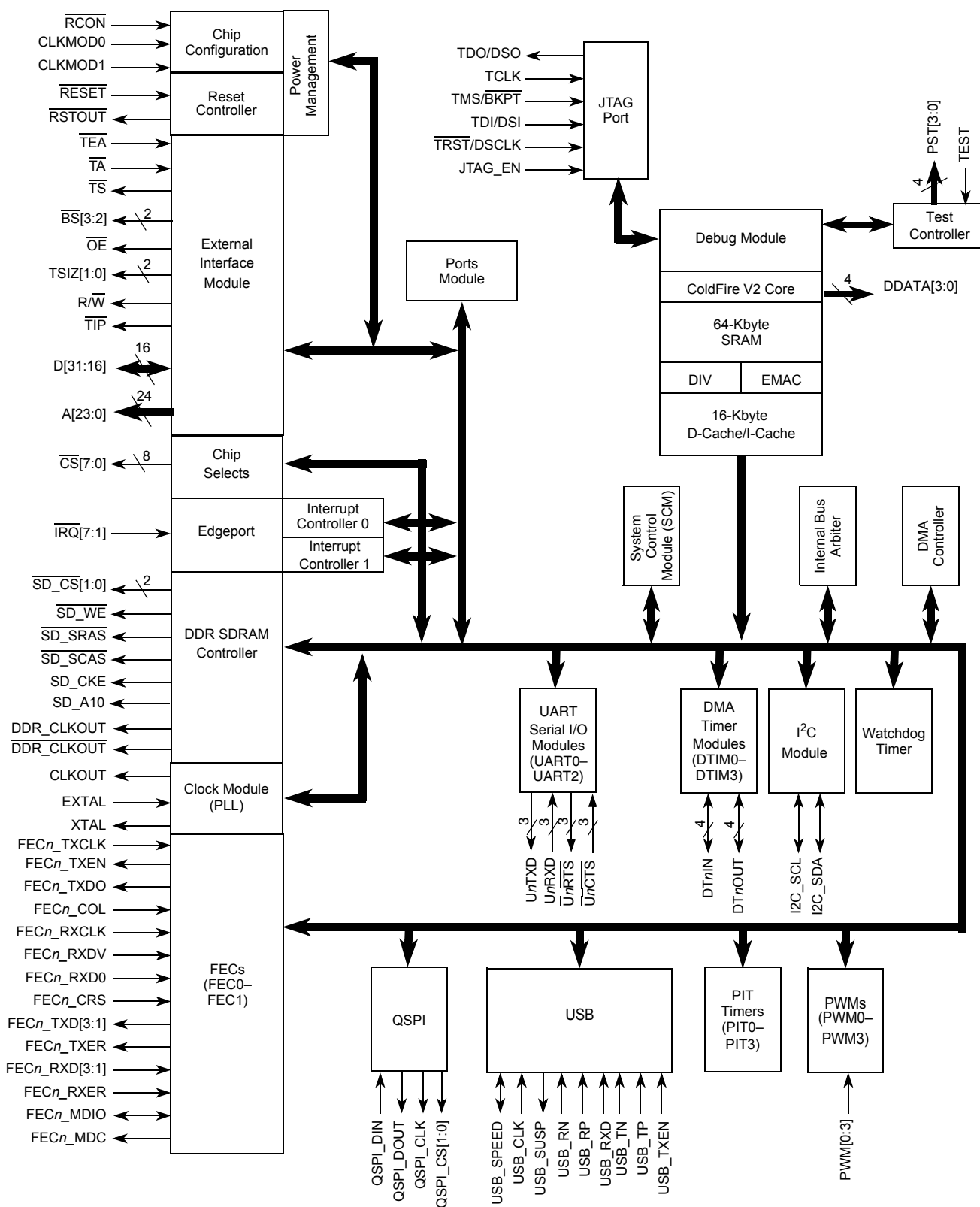
#### NOTE:

The terms ‘assertion’ and ‘negation’ are used to avoid confusion when dealing with a mixture of active-low and active-high signals. The term ‘asserted’ indicates that a signal is active, independent of the voltage level. The term ‘negated’ indicates that a signal is inactive.

Active-low signals, such as  $\overline{\text{SD\_SRAS}}$  and  $\overline{\text{TA}}$ , are indicated with an overbar.

#### 2.1.1 Overview

[Figure 2-1](#) shows the block diagram of the MCF5275 with the signal interface.



### Figure 2-1. MCF5275 Block Diagram with Signal Interfaces

## 2.2 Signal Properties Summary

Table 2-1 lists the MCF5275 signals grouped by functionality.

### NOTE:

In this table and throughout this document a single signal within a group is designated without square brackets (i.e., A24), while designations for multiple signals within a group use brackets (i.e., A[23:21]) and is meant to include all signals within the two bracketed numbers when these numbers are separated by a colon.

### NOTE:

The primary functionality of a pin is not necessarily its default functionality. Pins that are muxed with GPIO will default to their GPIO functionality.

**Table 2-1. MCF5274 and MCF5275 Signal Information and Muxing**

Signal Name	GPIO	Alternate1	Alternate2	Dir. <sup>1</sup>	MCF5274 MCF5275 256 MAPBGA	MCF5274L MCF5275L 196 MAPBGA
<b>Reset</b>						
$\overline{\text{RESET}}$	—	—	—	I	N15	K12
$\overline{\text{RSTOUT}}$	—	—	—	O	N14	L12
<b>Clock</b>						
EXTAL	—	—	—	I	L16	M14
XTAL	—	—	—	O	M16	N14
CLKOUT	—	—	—	O	T12	P9
<b>Mode Selection</b>						
CLKMOD[1:0]	—	—	—	I	N13, P13	M11, N11
$\overline{\text{RCON}}$	—	—	—	I	P8	M6
<b>External Memory Interface and Ports</b>						
A[23:21]	PADDR[7:5]	$\overline{\text{CS}}[6:4]$	—	O	A11, B11, C11	A8, B8, C8
A[20:0]	—	—	—	O	A12, B12, C12, A13, B13, C13, A14, B14, C14, B15, C15, B16, C16, D14, D15, E14:16, F14:16	B9, D9, C9, C10, B10, A11, C11, B11, A12, D11, C12, B13, C13, D12, E11, D13, E12, F11, D14, E13, F13

**Table 2-1. MCF5274 and MCF5275 Signal Information and Muxing (Continued)**

Signal Name	GPIO	Alternate1	Alternate2	Dir. <sup>1</sup>	MCF5274 MCF5275 256 MAPBGA	MCF5274L MCF5275L 196 MAPBGA
D[31:16]	—	—	—	O	M1, N1, N2, N3, P1, P2, R1, R2, P3, R3, T3, N4, P4, R4, T4, N5	J3, L1, K2, K3, M1, L2, L3, L4, K4, J4, M2, N1, N2, M3, M4, N3
$\overline{\text{BS}}[3:2]$	$\overline{\text{PBS}}[3:2]$	$\overline{\text{CAS}}[3:2]$	—	O	M3, R5	K1, L5
$\overline{\text{OE}}$	PBUSCTL[7]	—	—	O	K1	H4
$\overline{\text{TA}}$	PBUSCTL[6]	—	—	I	L13	K14
$\overline{\text{TEA}}$	PBUSCTL[5]	$\overline{\text{DREQ1}}$	—	I	T8	—
R/W	PBUSCTL[4]	—	—	O	P7	L6
TSIZ1	PBUSCTL[3]	DACK1	—	O	D16	B14
TSIZ0	PBUSCTL[2]	DACK0	—	O	G16	E14
$\overline{\text{TS}}$	PBUSCTL[1]	DACK2	—	O	L4	H2
$\overline{\text{TIP}}$	PBUSCTL[0]	$\overline{\text{DREQ0}}$	—	O	P6	—
<b>Chip Selects</b>						
$\overline{\text{CS}}[7:1]$	PCS[7:1]	—	—	O	D10:13, E13, F13, N7	D8, A9, A10, D10, B12, C14, P4
$\overline{\text{CS0}}$	—	—	—	O	R6	N5
<b>DDR SDRAM Controller</b>						
DDR_CLKOUT	—	—	—	O	T7	P6
$\overline{\text{DDR\_CLKOUT}}$	—	—	—	O	T6	P5
$\overline{\text{SD\_CS}}[1:0]$	PSDRAM[7:6]	$\overline{\text{CS}}[3:2]$	—	O	M2, T5	H3, M5
$\overline{\text{SD\_SRAS}}$	PSDRAM[5]	—	—	O	L2	H1
$\overline{\text{SD\_SCAS}}$	PSDRAM[4]	—	—	O	L1	G3
$\overline{\text{SD\_WE}}$	PSDRAM[3]	—	—	O	K2	G4
SD_A10	—	—	—	O	N6	N4
$\overline{\text{SD\_DQS}}[3:2]$	PSDRAM[2:1]	—	—	I/O	M4, P5	J2, P3
SD_CKE	PSDRAM[0]	—	—	O	L3	J1
SD_VREF	—	—	—	I	A15, T2	A13, P2
<b>External Interrupts Port</b>						
$\overline{\text{IRQ}}[7:5]$	PIRQ[7:5]	—	—	I	G13, H16, H15	F14, G13, G14
$\overline{\text{IRQ}}[4]$	PIRQ[4]	$\overline{\text{DREQ2}}$	—	I	H14	H11
$\overline{\text{IRQ}}[3:2]$	PIRQ[3:2]	$\overline{\text{DREQ}}[3:2]$	—	I	J14, J13	H14, H12



**Table 2-1. MCF5274 and MCF5275 Signal Information and Muxing (Continued)**

Signal Name	GPIO	Alternate1	Alternate2	Dir. <sup>1</sup>	MCF5274 MCF5275 256 MAPBGA	MCF5274L MCF5275L 196 MAPBGA
$\overline{\text{IRQ1}}$	PIRQ[1]	—	—	I	K13	J13
<b>FEC0</b>						
FEC0_MDIO	PFECI2C[5]	I2C_SDA	U2RXD	I/O	A7	A3
FEC0_MDC	PFECI2C[4]	I2C_SCL	U2TXD	O	B7	C5
FEC0_TXCLK	PFEC0H[7]	—	—	I	C3	C1
FEC0_TXEN	PFEC0H[6]	—	—	O	D4	C3
FEC0_TXD[0]	PFEC0H[5]	—	—	O	G4	D2
FEC0_COL	PFEC0H[4]	—	—	I	A6	B4
FEC0_RXCLK	PFEC0H[3]	—	—	I	B6	B3
FEC0_RXDV	PFEC0H[2]	—	—	I	B5	C4
FEC0_RXD[0]	PFEC0H[1]	—	—	I	C6	D5
FEC0_CRS	PFEC0H[0]	—	—	I	C7	A2
FEC0_TXD[3:1]	PFEC0L[7:5]	—	—	O	E3, F3, F4	D1, E3, D3
FEC0_TXER	PFEC0L[4]	—	—	O	D3	C2
FEC0_RXD[3:1]	PFEC0L[3:1]	—	—	I	D5, C5, D6	D4, B1, B2
FEC0_RXER	PFEC0L[0]	—	—	I	C4	E4
<b>FEC1</b>						
FEC1_MDIO	PFECI2C[3]	—	—	I/O	G1	—
FEC1_MDC	PFECI2C[2]	—	—	O	G2	—
FEC1_TXCLK	PFEC1H[7]	—	—	I	C1	—
FEC1_TXEN	PFEC1H[6]	—	—	O	D2	—
FEC1_TXD[0]	PFEC1H[5]	—	—	O	F1	—
FEC1_COL	PFEC1H[4]	—	—	I	A5	—
FEC1_RXCLK	PFEC1H[3]	—	—	I	B4	—
FEC1_RXDV	PFEC1H[2]	—	—	I	A3	—
FEC1_RXD[0]	PFEC1H[1]	—	—	I	B3	—
FEC1_CRS	PFEC1H[0]	—	—	I	A4	—
FEC1_TXD[3:1]	PFEC1L[7:5]	—	—	O	E1, E2, F2	—
FEC1_TXER	PFEC1L[4]	—	—	O	D1	—
FEC1_RXD[3:1]	PFEC1L[3:1]	—	—	I	B1, B2, A2	—
FEC1_RXER	PFEC1L[0]	—	—	I	C2	—

**Table 2-1. MCF5274 and MCF5275 Signal Information and Muxing (Continued)**

Signal Name	GPIO	Alternate1	Alternate2	Dir. <sup>1</sup>	MCF5274 MCF5275 256 MAPBGA	MCF5274L MCF5275L 196 MAPBGA
<b>I<sup>2</sup>C</b>						
I2C_SDA	PFECI2C[1]	U2RXD	—	I/O	B10	B7
I2C_SCL	PFECI2C[0]	U2TXD	—	I/O	C10	A7
<b>DMA</b>						
DACK[3:0] and DREQ[3:0] do not have a dedicated bond pads. Please refer to the following pins for muxing: PCS3/PWM3 for DACK3, PCS2/PWM2 for DACK2, TSIZ1 for DACK1, TSIZ0 for DACK0, IRQ3 for DREQ3, IRQ2 and TA for DREQ2, TEA for DREQ1, and TIP for DREQ0.					—	—
<b>QSPI</b>						
QSPI_CS[3:2]	PQSPI[6:5]	PWM[3:2]	DACK[3:2]	O	R13, N12	P10, N9
QSPI_CS1	PQSPI[4]	—	—	O	T14	N10
QSPI_CS0	PQSPI[3]	—	—	O	P12	M9
QSPI_CLK	PQSPI[2]	I2C_SCL	—	O	T15	L11
QSPI_DIN	PQSPI[1]	I2C_SDA	—	I	T13	M10
QSPI_DOUT	PQSPI[0]	—	—	O	R12	L10
<b>UARTs</b>						
U2RXD	PUARTH[3]	—	—	I	T9	—
U2TXD	PUARTH[2]	—	—	O	R9	—
$\overline{\text{U2CTS}}$	PUARTH[1]	PWM1	—	I	P9	—
$\overline{\text{U2RTS}}$	PUARTH[0]	PWM0	—	O	R8	—
U1RXD	PUARTL[7]	—	—	I	A9	A6
U1TXD	PUARTL[6]	—	—	O	B9	D7
$\overline{\text{U1CTS}}$	PUARTL[5]	—	—	I	C9	C7
$\overline{\text{U1RTS}}$	PUARTL[4]	—	—	O	D9	B6
U0RXD	PUARTL[3]	—	—	I	A8	A4
U0TXD	PUARTL[2]	—	—	O	B8	A5
$\overline{\text{U0CTS}}$	PUARTL[1]	—	—	I	C8	C6
$\overline{\text{U0RTS}}$	PUARTL[0]	—	—	O	D7	B5
<b>USB</b>						
USB_SPEED	PUSBH[0]	—	—	I/O	G14	G11
USB_CLK	PUSBL[7]	—	—	I	G15	F12

**Table 2-1. MCF5274 and MCF5275 Signal Information and Muxing (Continued)**

Signal Name	GPIO	Alternate1	Alternate2	Dir. <sup>1</sup>	MCF5274 MCF5275 256 MAPBGA	MCF5274L MCF5275L 196 MAPBGA
USB_RN	PUSBL[6]	—	—	I	J16	H13
USB_RP	PUSBL[5]	—	—	I	J15	J11
USB_RXD	PUSBL[4]	—	—	I	L15	L14
USB_SUSP	PUSBL[3]	—	—	O	M13	N13
USB_TN	PUSBL[2]	—	—	O	K14	J14
USB_TP	PUSBL[1]	—	—	O	K15	J12
USB_TXEN	PUSBL[0]	—	—	O	L14	K13
<b>Timers (and PWMs)</b>						
DT3IN	PTIMERH[3]	DT3OUT	$\overline{U2RTS}$	I	J4	G2
DT3OUT	PTIMERH[2]	PWM3	$\overline{U2CTS}$	O	K3	G1
DT2IN	PTIMERH[1]	DT2OUT	—	I	J2	F3
DT2OUT	PTIMERH[0]	PWM2	—	O	J3	F4
DT1IN	PTIMERL[3]	DT1OUT	—	I	H1	F1
DT1OUT	PTIMERL[2]	PWM1	—	O	H2	F2
DT0IN	PTIMERL[1]	DT0OUT	—	I	H3	E1
DT0OUT	PTIMERL[0]	PWM0	—	O	G3	E2
<b>BDM/JTAG<sup>2</sup></b>						
DSCLK	—	$\overline{TRST}$	—	I	P14	P13
PSTCLK	—	TCLK	—	O	P16	P12
$\overline{BKPT}$	—	TMS	—	I	R15	N12
DSI	—	TDI	—	I	R16	M12
DSO	—	TDO	—	O	P15	K11
JTAG_EN	—	—	—	I	R14	P11
DDATA[3:0]	—	—	—	O	P10, N10, P11, N11	M7, N7, P8, L9
PST[3:0]	—	—	—	O	T10, R10, T11, R11	P7, L8, M8, N8
<b>Test</b>						
TEST	—	—	—	I	N9	N6
PLL_TEST	—	—	—	I	M14	—
<b>Power Supplies</b>						

**Table 2-1. MCF5274 and MCF5275 Signal Information and Muxing (Continued)**

Signal Name	GPIO	Alternate1	Alternate2	Dir. <sup>1</sup>	MCF5274 MCF5275 256 MAPBGA	MCF5274L MCF5275L 196 MAPBGA
VDDPLL	—	—	—	I	M15	M13
VSSPLL	—	—	—	I	K16	L13
VSS	—	—	—	I	A1, A10, A16, E5, E12, F6, F11, G7:10, H7:10, J1, J7:10, K7:10, L6, L11, M5, N16, R7, T1, T16	F7, F8, G6:9, H6:9, J7, J8
OVDD	—	—	—	I	E6:8, F5, F7, F8, G5, G6, H5, H6, J11, J12, K11, K12, L9, L10, L12, M9:11	E5:7, F5, F6, H10, J9, J10, K8:10
VDD	—	—	—	I	D8, H13, K4, N8	D6, G5, G12, L7
SD_VDD	—	—	—	I	E9:11, F9, F10, F12, G11, G12, H11, H12, J5, J6, K5, K6, L5, L7, L8, M6, M7, M8	E8:10, F9, F10, G10, H5, J5, J6, K5:7

<sup>1</sup> Refers to pin's primary function. All pins which are configurable for GPIO have a pullup enabled in GPIO mode with the exception of PBUSCTL[7], PBUSCTL[4:0], PADDR, PBS, PSDRAM.

<sup>2</sup> If JTAG\_EN is asserted, these pins default to Alternate 1 (JTAG) functionality. The GPIO module is not responsible for assigning these pins.

## 2.3 External Signal Descriptions

### 2.3.1 Reset Signals

Table 2-2 describes signals that are used to either reset the chip or as a reset indication.

**Table 2-2. Reset Signals**

Signal Name	Abbreviation	Function	I/O
Reset In	$\overline{\text{RESET}}$	Primary reset input to the device. Asserting $\overline{\text{RESET}}$ immediately resets the CPU and peripherals.	I
Reset Out	$\overline{\text{RSTOUT}}$	Driven low for 128 CPU clocks when the soft reset bit of the system configuration register (SCR[SOFTRST]) is set. It is driven low for 32K CPU clocks when the software watchdog timer times out or when a low input level is applied to $\overline{\text{RESET}}$ .	O

## 2.3.2 PLL and Clock Signals

Table 2-3 describes signals that are used to support the on-chip clock generation circuitry.

**Table 2-3. PLL and Clock Signals**

Signal Name	Abbreviation	Function	I/O
External Clock In	EXTAL	Always driven by an external clock input except when used as a connection to the external crystal when the internal oscillator circuit is used. The clock source is configured during reset by CLKMOD[1:0].	I
Crystal	XTAL	Used as a connection to the external crystal when the internal oscillator circuit is used to drive the crystal.	O
Clock Out	CLKOUT	This output signal reflects the internal system clock.	O

## 2.3.3 Mode Selection

Table 2-4 describes signals used in mode selection.

**Table 2-4. Mode Selection Signals**

Signal Name	Abbreviation	Function	I/O
Clock Mode Selection	CLKMOD[1:0]	Configure the clock mode after reset.	I
Reset Configuration	RCON	Indicates whether the external D[31:16] pin states affect chip configuration at reset.	I

## 2.3.4 External Memory Interface Signals

These signals are used for doing transactions on the external bus.

Table 2-5 describes signals that are used for doing transactions on the external bus.

**Table 2-5. External Memory Interface Signals**

Signal Name	Abbreviation	Function	I/O
Address Bus	A[23:0]	The 24 dedicated address signals define the address of external byte, word, and longword accesses. These three-state outputs are the 24 lsbs of the internal 32-bit address bus and multiplexed with the SDRAM controller row and column addresses.	O
Data Bus	D[31:16]	These three-state bidirectional signals provide the general purpose data path between the processor and all other devices.	I/O

**Table 2-5. External Memory Interface Signals (Continued)**

Signal Name	Abbreviation	Function	I/O
Byte Strokes	$\overline{BS}[3:2]$	<p>Define the flow of data on the data bus. During SRAM and peripheral accesses, these output signals indicate that data is to be latched or driven onto a byte of the data when driven low. The <math>\overline{BS}[3:2]</math> signals are asserted only to the memory bytes used during a read or write access. <math>\overline{BS2}</math> controls access to the least significant byte lane of data, and <math>\overline{BS3}</math> controls access to the most significant byte lane of data.</p> <p>The <math>\overline{BS}[3:2]</math> signals are asserted during accesses to on-chip peripherals but not to on-chip SRAM, or cache. During SDRAM accesses, these signals act as the <math>\overline{CAS}[3:2]</math> signals, which indicate a byte transfers between SDRAM and the chip when driven high.</p> <p>For SRAM or Flash devices, the <math>\overline{BS}[3:2]</math> outputs should be connected to individual byte strobe signals.</p> <p>For SDRAM devices, the <math>\overline{BS}[3:2]</math> should be connected to individual SDRAM DQM signals. Note that most SDRAMs associate DQM1 with the MSB, in which case <math>\overline{BS3}</math> should be connected to the SDRAM's DQM1 input.</p>	O
Output Enable	$\overline{OE}$	Indicates when an external device can drive data during external read cycles.	O
Transfer Acknowledge	$\overline{TA}$	Indicates that the external data transfer is complete. During a read cycle, when the processor recognizes $\overline{TA}$ , it latches the data and then terminates the bus cycle. During a write cycle, when the processor recognizes $\overline{TA}$ , the bus cycle is terminated.	I
Transfer Error Acknowledge	$\overline{TEA}$	Indicates an error condition exists for the bus transfer. The bus cycle is terminated and the CPU begins execution of the access error exception.	I
Read/Write	$R/\overline{W}$	Indicates the direction of the data transfer on the bus for SRAM ( $R/\overline{W}$ ) and SDRAM ( $\overline{SD\_WE}$ ) accesses. A logic 1 indicates a read from a slave device and a logic 0 indicates a write to a slave device	O
Transfer Size	$\overline{TSIZ}[1:0]$	When the device is in normal mode, dynamic bus sizing lets the programmer change data bus width between 8, 16, and 32 bits for each chip select. The initial width for the bootstrap program chip select, CS0, is determined by the state of $\overline{TSIZ}[1:0]$ . The program should select bus widths for the other chip selects before accessing the associated memory space. These pins are output pins.	O
Transfer Start	$\overline{TS}$	Bus control output signal indicating the start of a transfer.	O
Transfer in Progress	$\overline{TIP}$	Bus control output signal indicating bus transfer in progress.	O
Chip Selects	$\overline{CS}[7:0]$	These output signals select external devices for external bus transactions. The $\overline{CS}[3:2]$ can also be configured to function as SDRAM chip selects $\overline{SD\_CS}[1:0]$ .	O

## 2.3.5 DDR SDRAM Controller Signals

Table 6 describes signals that are used for DDR SDRAM accesses.

**Table 6. SDRAM Controller Signals**

Signal Name	Abbreviation	Function	I/O
SDRAM Clock Out	DDR_CLKOUT	This output signal reflects the internal system clock.	O
SDRAM Inverted Clock Out	$\overline{\text{DDR\_CLKOUT}}$	This output signal reflects the inverted internal system clock.	O
SDRAM Synchronous Row Address Strobe	$\overline{\text{SD\_SRAS}}$	SDRAM synchronous row address strobe.	O
SDRAM Synchronous Column Address Strobe	$\overline{\text{SD\_SCAS}}$	SDRAM synchronous column address strobe.	O
SDRAM Write Enable	$\overline{\text{SD\_WE}}$	SDRAM write enable.	O
SDRAM A10	SD_A10	SDRAM address bit 10 or command.	O
SDRAM Chip Selects	$\overline{\text{SD\_CS}}[1:0]$	SDRAM chip select signals.	O
SDRAM Clock Enable	$\overline{\text{SD\_CKE}}$	SDRAM clock enable.	O
SDRAM Data Strobes	$\overline{\text{SD\_DQS}}[3:2]$	SDRAM byte-lane read/write data strobe signals.	O

## 2.3.6 External Interrupt Signals

Table 2-7 describes the external interrupt signals.

**Table 2-7. External Interrupt Signals**

Signal Name	Abbreviation	Function	I/O
External Interrupts	$\overline{\text{IRQ}}[7:1]$	External interrupt sources. $\overline{\text{IRQ}}[3:2]$ can also be configured as DMA request signals $\overline{\text{DREQ}}[3:2]$ . $\overline{\text{IRQ}}4$ can also be configured as DMA request signals $\overline{\text{DREQ}}2$ .	I

## 2.3.7 Fast Ethernet Controller Signals

The following signals are used by the Ethernet modules for data and clock signals.

**Table 2-8. Ethernet Module (FEC) Signals**

Signal Name	Abbreviation	Function	I/O
Management Data	FEC <sub>n</sub> _MDIO	Transfers control information between the external PHY and the media-access controller. Data is synchronous to FEC <sub>n</sub> _MDC. Applies to MII mode operation. This signal is an input after reset. When the FEC is operated in 10Mbps 7-wire interface mode, this signal should be connected to VSS.	I/O
Management Data Clock	FEC <sub>n</sub> _MDC	In Ethernet mode, FEC <sub>n</sub> _MDC is an output clock which provides a timing reference to the PHY for data transfers on the FEC <sub>n</sub> _MDIO signal. Applies to MII mode operation.	O
Transmit Clock	FEC <sub>n</sub> _TXCLK	Input clock which provides a timing reference for FEC <sub>n</sub> _TXEN, FEC <sub>n</sub> _TXD[3:0] and FEC <sub>n</sub> _TXER	I

**Table 2-8. Ethernet Module (FEC) Signals (Continued)**

Signal Name	Abbreviation	Function	I/O
Transmit Enable	FEC <sub>n</sub> _TXEN	Indicates when valid nibbles are present on the MII. This signal is asserted with the first nibble of a preamble and is negated before the first FEC <sub>n</sub> _TXCLK following the final nibble of the frame.	O
Transmit Data 0	FEC <sub>n</sub> _TXD0	FEC <sub>n</sub> _TXD0 is the serial output Ethernet data and is only valid during the assertion of FEC <sub>n</sub> _TXEN. This signal is used for 10-Mbps Ethernet data. It is also used for MII mode data in conjunction with FEC <sub>n</sub> _TXD[3:1].	O
Collision	FEC <sub>n</sub> _COL	Asserted upon detection of a collision and remains asserted while the collision persists. This signal is not defined for full-duplex mode.	I
Receive Clock	FEC <sub>n</sub> _RXCLK	Provides a timing reference for FEC <sub>n</sub> _RXDV, FEC <sub>n</sub> _RXD[3:0], and FEC <sub>n</sub> _RXER.	I
Receive Data Valid	FEC <sub>n</sub> _RXDV	Asserting the receive data valid (FEC <sub>n</sub> _RXDV) input indicates that the PHY has valid nibbles present on the MII. FEC <sub>n</sub> _RXDV should remain asserted from the first recovered nibble of the frame through to the last nibble. Assertion of FEC <sub>n</sub> _RXDV must start no later than the SFD and exclude any EOF.	I
Receive Data 0	FEC <sub>n</sub> _RXD0	FEC <sub>n</sub> _RXD0 is the Ethernet input data transferred from the PHY to the media-access controller when FEC <sub>n</sub> _RXDV is asserted. This signal is used for 10-Mbps Ethernet data. This signal is also used for MII mode Ethernet data in conjunction with FEC <sub>n</sub> _RXD[3:1].	I
Carrier Receive Sense	FEC <sub>n</sub> _CRS	When asserted, indicates that transmit or receive medium is not idle. Applies to MII mode operation.	I
Transmit Data 1–3	FEC <sub>n</sub> _TXD[3:1]	In Ethernet mode, these pins contain the serial output Ethernet data and are valid only during assertion of FEC <sub>n</sub> _TXEN in MII mode.	O
Transmit Error	FEC <sub>n</sub> _TXER	In Ethernet mode, when FEC <sub>n</sub> _TXER is asserted for one or more clock cycles while FEC <sub>n</sub> _TXEN is also asserted, the PHY sends one or more illegal symbols. FEC <sub>n</sub> _TXER has no effect at 10 Mbps or when FEC <sub>n</sub> _TXEN is negated. Applies to MII mode operation.	O
Receive Data 1–3	FEC <sub>n</sub> _RXD[3:1]	In Ethernet mode, these pins contain the Ethernet input data transferred from the PHY to the Media Access Controller when FEC <sub>n</sub> _RXDV is asserted in MII mode operation.	I
Receive Error	FEC <sub>n</sub> _RXER	In Ethernet mode, FEC <sub>n</sub> _RXER—when asserted with FEC <sub>n</sub> _RXDV—indicates that the PHY has detected an error in the current frame. When FEC <sub>n</sub> _RXDV is not asserted FEC <sub>n</sub> _RXER has no effect. Applies to MII mode operation.	I

## 2.3.8 Queued Serial Peripheral Interface (QSPI)

Table 2-9 describes QSPI signals.



**Table 2-9. Queued Serial Peripheral Interface (QSPI) Signals**

Signal Name	Abbreviation	Function	I/O
QSPI Synchronous Serial Output	QSPI_DOUT	Provides the serial data from the QSPI and can be programmed to be driven on the rising or falling edge of QSPI_CLK. Each byte is sent msb first.	O
QSPI Synchronous Serial Data Input	QSPI_DIN	Provides the serial data to the QSPI and can be programmed to be sampled on the rising or falling edge of QSPI_CLK. Each byte is written to RAM lsb first.	I
QSPI Serial Clock	QSPI_CLK	Provides the serial clock from the QSPI. The polarity and phase of QSPI_CLK are programmable. The output frequency is programmed according to the following formula, in which $n$ can be any value between 1 and 255: $\text{SPI\_CLK} = f_{\text{sys}}/2 \div n$	O
Synchronous Peripheral Chip Selects	QSPI_CS[1:0]	Provide QSPI peripheral chip selects that can be programmed to be active high or low. QSPI_CS1 can also be configured as SDRAM clock enable signal SD_CKE.	O

## 2.3.9 I<sup>2</sup>C I/O SIGNALS

Table 2-10 describes the I<sup>2</sup>C serial interface module signals.

**Table 2-10. I<sup>2</sup>C I/O Signals**

Signal Name	Abbreviation	Function	I/O
Serial Clock	I2C_SCL	Open-drain clock signal for the for the I <sup>2</sup> C interface. Either it is driven by the I <sup>2</sup> C module when the bus is in the master mode or it becomes the clock input when the I <sup>2</sup> C is in the slave mode.	I/O
Serial Data	I2C_SDA	Open-drain signal that serves as the data input/output for the I <sup>2</sup> C interface.	I/O

## 2.3.10 UART Module Signals

The UART modules use the signals in this section for data. The baud rate clock inputs are not supported.

**Table 2-11. UART Module Signals**

Signal Name	Abbreviation	Function	I/O
Transmit Serial Data Output	UnTXD	Transmitter serial data outputs for the UART modules. The output is held high (mark condition) when the transmitter is disabled, idle, or in the local loopback mode. Data is shifted out, lsb first, on this pin at the falling edge of the serial clock source.	O
Receive Serial Data Input	UnRXD	Receiver serial data inputs for the UART modules. Data received on this pin is sampled on the rising edge of the serial clock source lsb first. When the UART clock is stopped for power-down mode, any transition on this pin restarts it.	I

**Table 2-11. UART Module Signals (Continued)**

Signal Name	Abbreviation	Function	I/O
Clear-to-Send	$\overline{U}nCTS$	Indicate to the UART modules that they can begin data transmission.	I
Request-to-Send	$\overline{U}nRTS$	Automatic request-to-send outputs from the UART modules. $\overline{U}nRTS$ can also be configured to be asserted and negated as a function of the RxFIFO level.	O

## 2.3.11 USB Signals

Table 2-12 describes the USB serial interface module signals.

**Table 2-12. USB Module Signals**

Signal Name	Abbreviation	Function	I/O
USB Clock	USB_CLK	This 48MHz (or 6MHz) clock is used by the USB module for both clock recovery and generation of a 12Mhz (or 1.5MHz) internal bit clock.	I
USB Speed	USB_SPEED	Applications which make use of low speed USB signalling must be able to switch the USB transceiver between low speed and full speed operations. Software has control of this function by driving the state of the USB_SPD bit in the USB_CTRL register onto the USB_SPEED pin.	I/O
USB Received D-	USB_RN	This signal is one half of the differential USB signal, and is extracted from the USB cable via a single ended input buffer on the analog front end. This signal is used by the module for detecting the single ended 0 (SE0) USB bus state.	I
USB Received D+	USB_RP	This signal is one half of the differential USB signal, and is extracted from the USB cable via a single ended input buffer on the analog front end. This signal is used by the module for detecting the single ended 0 (SE0) USB bus state.	I
USB Receive Data	USB_RXD	Input data from the differential input receiver. USB_RXD is the single-ended data extracted from the USB_RP and USB_RN signals via a differential input buffer.	I
USB Suspended	USB_SUSP	After a long period of inactivity (3.0ms minimum), the USB will enter suspend mode, indicated on the interface by an active state on USB_SUSP. During this mode, the device is supposed to enter a low power state while waiting for a wake-up from the USB Host. When the device enters suspend mode, it asserts the suspend signal which forces the analog front end into a low power state. When the device leaves suspend mode, USB_SUSP is deasserted, enabling the analog front end for normal USB operations.	O
USB Transmitted D-	USB_TN	This signal is one half of the differential NRZI formatted output from the USB module. It is fed to the transmitted D- input of the analog front end.	O

**Table 2-12. USB Module Signals (Continued)**

Signal Name	Abbreviation	Function	I/O
USB Transmitted D+	USB_TP	This signal is one half of the differential NRZI formatted output from the module. It is fed to the transmitted D+ input of the analog front end.	O
USB Transmit Enable	USB_TXEN	This signal is an active low output enable for the differential drivers on the analog front end. When this signal is active, the differential drivers will drive the USB. When this signal is inactive, the differential drivers will tristate their outputs.	O

## 2.3.12 DMA Timer Signals

Table 2-13 describes the signals of the four DMA timer modules.

**Table 2-13. DMA Timer Signals**

Signal Name	Abbreviation	Function	I/O
DMA Timer 0 Input	DT0IN	Can be programmed to cause events to occur in first platform timer. It can either clock the event counter or provide a trigger to the timer value capture logic.	I
DMA Timer 0 Output	DT0OUT	The output from first platform timer.	O
DMA Timer 1 Input	DT1IN	Can be programmed to cause events to occur in the second platform timer. This can either clock the event counter or provide a trigger to the timer value capture logic.	I
DMA Timer 1 Output	DT1OUT	The output from the second platform timer.	O
DMA Timer 2 Input	DT2IN	Can be programmed to cause events to occur in the third platform timer. It can either clock the event counter or provide a trigger to the timer value capture logic.	I
DMA Timer 2 Output	DT2OUT	The output from the third platform timer.	I
DMA Timer 3 Input	DT3IN	Can be programmed as an input that causes events to occur in the fourth platform timer. This can either clock the event counter or provide a trigger to the timer value capture logic.	I
DMA Timer 3 Output	DT3OUT	The output from the fourth platform timer.	O

## 2.3.13 Pulse Width Modulator Signals

Table 2-14 describes the PWM signals. Note that the primary functions of these pins are DMA Timer outputs (DT $n$ OUT).

**Table 2-14. PWM Signals**

Signal Name	Abbreviation	Function	I/O
PWM Output Channel 0	PWM0	Pulse width modulated output for PWM channel 0.	O
PWM Output Channel 1	PWM1	Pulse width modulated output for PWM channel 1.	O
PWM Output Channel 2	PWM2	Pulse width modulated output for PWM channel 2.	O
PWM Output Channel 3	PWM3	Pulse width modulated output for PWM channel 3.	O

## 2.3.14 Debug Support Signals

These signals are used as the interface to the on-chip JTAG controller and also to interface to the BDM logic.

**Table 2-15. Debug Support Signals**

Signal Name	Abbreviation	Function	I/O
Test Reset	$\overline{\text{TRST}}$	This active-low signal is used to initialize the JTAG logic asynchronously.	I
Test Clock	TCLK	Used to synchronize the JTAG logic.	I
Test Mode Select	TMS	Used to sequence the JTAG state machine. TMS is sampled on the rising edge of TCLK.	I
Test Data Input	TDI	Serial input for test instructions and data. TDI is sampled on the rising edge of TCLK.	I
Test Data Output	TDO	Serial output for test instructions and data. TDO is three-stateable and is actively driven in the shift-IR and shift-DR controller states. TDO changes on the falling edge of TCLK.	O
Development Serial Clock	DSCLK	Clocks the serial communication port to the BDM module during packet transfers.	I
Breakpoint	$\overline{\text{BKPT}}$	Used to request a manual breakpoint.	I
Development Serial Input	DSI	This internally-synchronized signal provides data input for the serial communication port to the BDM module.	I
Development Serial Output	DSO	This internally-registered signal provides serial output communication for BDM module responses.	O
Debug Data	DDATA[3:0]	Display captured processor data and breakpoint status. The CLKOUT signal can be used by the development system to know when to sample DDATA[3:0].	O
Processor Status Outputs	PST[3:0]	Indicate core status, as shown in <a href="#">Table 2-16</a> . Debug mode timing is synchronous with the processor clock; status is unrelated to the current bus transfer. The CLKOUT signal can be used by the development system to know when to sample PST[3:0].	O
Processor Status Clock	PSTCLK	PSTCLK indicates when the development systems should sample PST and DDATA values.	O

**Table 2-16. Processor Status**

PST[3:0]	Processor Status
0000	Continue execution
0001	Begin execution of one instruction
0010	Reserved
0011	Entry into user mode
0100	Begin execution of PULSE and WDDATA instructions
0101	Begin execution of taken branch
0110	Reserved
0111	Begin execution of RTE instruction
1000	Begin one-byte transfer on DDATA
1001	Begin two-byte transfer on DDATA
1010	Begin three-byte transfer on DDATA
1011	Begin four-byte transfer on DDATA
1100	Exception processing
1101	Reserved
1110	Processor is stopped
1111	Processor is halted

## 2.3.15 Test Signals

Table 2-17 describes test signals.

**Table 2-17. Test Signals**

Signal Name	Abbreviation	Function	I/O
Test	TEST	Reserved for factory testing only and in normal modes of operation should be connected to VSS to prevent unintentional activation of test functions.	I
PLL Test	PLL_TEST	Reserved for factory testing only and should be treated as a no-connect (NC).	I

## 2.3.16 Power and Ground Pins

The pins described in Table 2-18 provide system power and ground to the chip. Multiple pins are provided for adequate current capability. All power supply pins must have adequate bypass capacitance for high-frequency noise suppression.

**Table 2-18. Power and Ground Pins**

Signal Name	Abbreviation	Function	I/O
PLL Analog Supply	VDDPLL, VSSPLL	Dedicated power supply signals to isolate the sensitive PLL analog circuitry from the normal levels of noise present on the digital power supply.	I
Positive Supply	VDDO	These pins supply positive power to the I/O pads.	I
Positive Supply	VDD	These pins supply positive power to the core logic.	I
Ground	VSS	This pin is the negative supply (ground) to the chip.	

## 2.4 External Boot Mode

When booting from external memory, the address bus, data bus, and bus control signals will default to their bus functionalities as shown in [Table 2-19](#). All other signals will default to GPIO inputs.

**Table 2-19. Default Signal Functions After System Reset (External Boot Mode)**

Signal	Reset	I/O
A[23:0]	A[23:0]	O
D[31:16]	—	I/O
$\overline{\text{BS}}[3:2]$	High	O
$\overline{\text{OE}}$	High	O
$\overline{\text{TA}}$	—	I
$\overline{\text{TEA}}$	—	I
$\text{R}/\overline{\text{W}}$	High	O
SIZ[1:0]	High	O
$\overline{\text{TS}}$	High	O
$\overline{\text{TIP}}$	High	O
$\overline{\text{CS}}[7:0]$	High	O

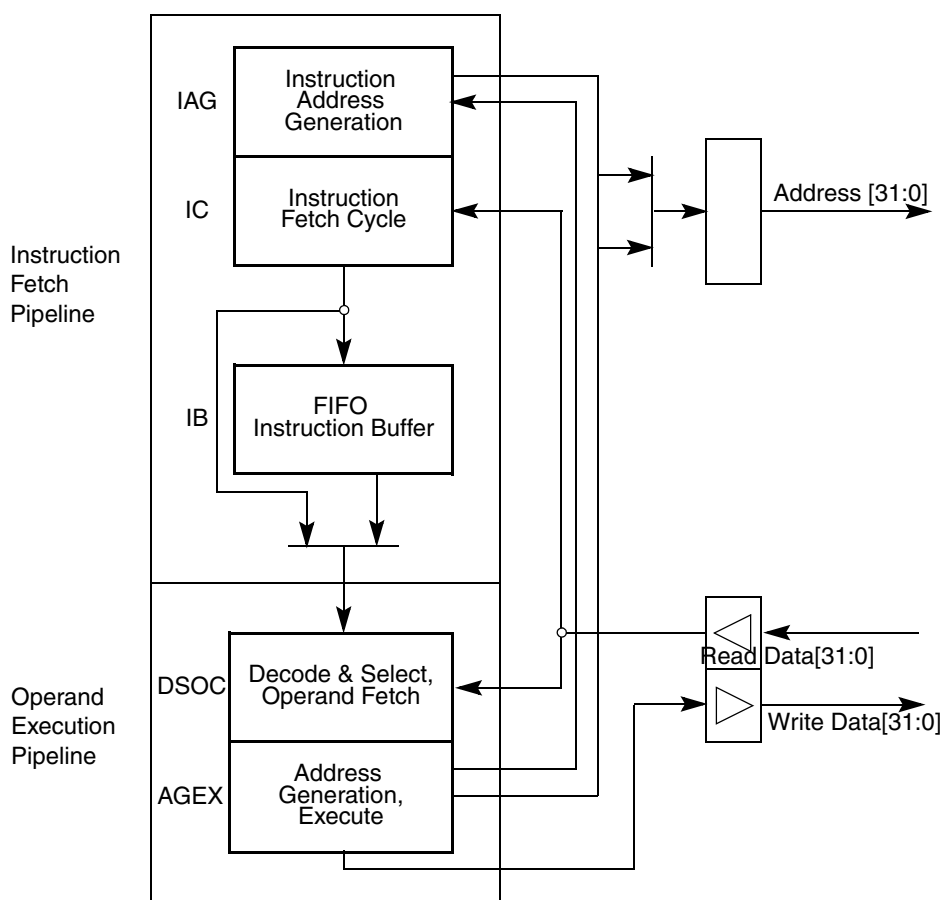
# Chapter 3

## ColdFire Core

This section describes the organization of the Version 2 (V2) ColdFire® processor core and an overview of the program-visible registers. For detailed information on instructions, see the *ColdFire Family Programmer's Reference Manual*.

### 3.1 Processor Pipelines

Figure 3-1 is a block diagram showing the processor pipelines of a V2 ColdFire core.



**Figure 3-1. ColdFire Processor Core Pipelines**

The processor core is comprised of two separate pipelines that are decoupled by an instruction buffer.

The Instruction Fetch Pipeline (IFP) is a two-stage pipeline for prefetching instructions. The prefetched instruction stream is then gated into the two-stage Operand Execution Pipeline (OEP),

which decodes the instruction, fetches the required operands and then executes the required function. Since the IFP and OEP pipelines are decoupled by an instruction buffer which serves as a FIFO queue, the IFP is able to prefetch instructions in advance of their actual use by the OEP thereby minimizing time stalled waiting for instructions.

The Instruction Fetch Pipeline consists of two stages with an instruction buffer stage:

- Instruction Address Generation (IAG Cycle)
- Instruction Fetch Cycle (IC Cycle)
- Instruction Buffer (IB Cycle)

When the instruction buffer is empty, opcodes are loaded directly from the IC cycle into the Operand Execution Pipeline. If the buffer is not empty, the IFP stores the contents of the fetch cycle in the FIFO queue until it is required by the OEP. In the Version 2 implementation, the instruction buffer contains three 32-bit longwords of storage.

The Operand Execution Pipeline is implemented in a two-stage pipeline featuring a traditional RISC datapath with a dual-read-ported register file (RGF) feeding an arithmetic/logic unit. In this design, the pipeline stages have multiple functions:

- Decode & Select/Operand Cycle (DSOC Cycle)
- Address Generation/Execute Cycle (AGEX Cycle)

## 3.2 Processor Register Description

The following paragraphs describe the processor registers in the user and supervisor programming models. The appropriate programming model is selected based on the privilege level (user mode or supervisor mode) of the processor as defined by the S bit of the status register (SR).

### 3.2.1 User Programming Model

Figure 3-2 illustrates the user programming model. The model is the same as the M68000 family microprocessors, consisting of the following registers:

- 16 general-purpose 32-bit registers (D0–D7, A0–A7)
- 32-bit program counter (PC)
- 8-bit condition code register (CCR)

#### 3.2.1.1 Data Registers (D0–D7)

Registers D0–D7 are used as data registers for bit (1-bit), byte (8-bit), word (16-bit) and longword (32-bit) operations; they can also be used as index registers.



### 3.2.1.2 Address Registers (A0–A6)

These registers can be used as software stack pointers, index registers, or base address registers; they can also be used for word and longword operations.

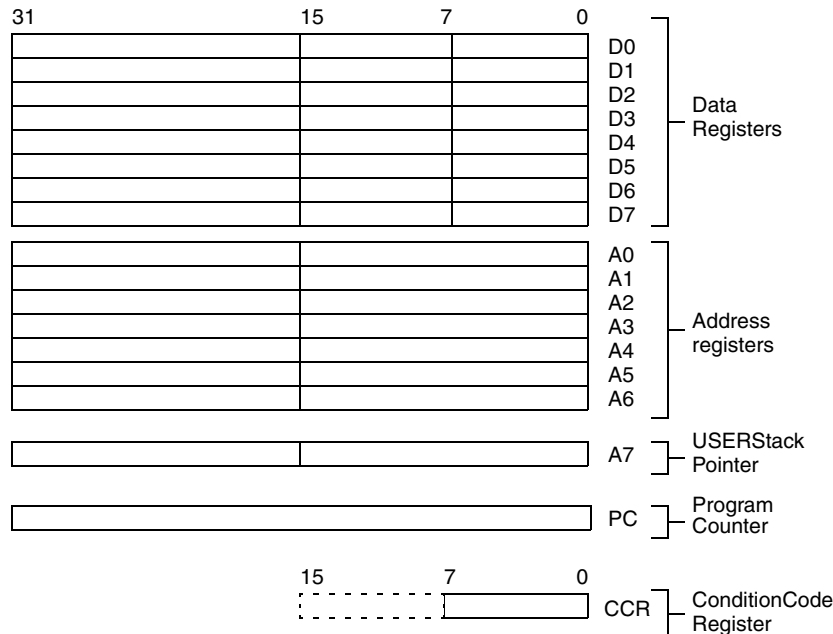
### 3.2.1.3 Stack Pointer (A7)

Certain ColdFire implementations, including the MCF5275, support two unique stack pointer (A7) registers—the supervisor stack pointer (SSP) and the user stack pointer (USP). This support provides the required isolation between operating modes of the processor. The SSP is described in [Section 3.2.3.2, “Supervisor/User Stack Pointers \(A7 and OTHER\\_A7\).”](#)

A subroutine call saves the PC on the stack and the return restores it from the stack. Both the PC and the SR are saved on the supervisor stack during the processing of exceptions and interrupts. The return from exception (RTE) instruction restores the SR and PC values from the supervisor stack.

### 3.2.1.4 Program Counter (PC)

The PC contains the address of the currently executing instruction. During instruction execution and exception processing, the processor automatically increments the contents of the PC or places a new value in the PC, as appropriate. For some addressing modes, the PC is used as a base address for PC-relative operand addressing.



**Figure 3-2. User Programming Model**

### 3.2.1.5 Condition Code Register (CCR)

The CCR is the LSB of the processor status register (SR). Bits 4–0 act as indicator flags for results generated by processor operations. Bit 4, the extend bit (X bit), is also used as an input operand during multiprecision arithmetic computations.

	7	6	5	4	3	2	1	0
Field	0	0	0	X	N	Z	V	C
Reset	0	0	0	0	0	0	0	0
Address	LSB of Status Register (SR)							

**Figure 3-3. Condition Code Register (CCR)**

**Table 3-1. CCR Field Descriptions**

Bits	Name	Description
7–5	—	Reserved, should be cleared
4	X	Extend condition code bit. Set to the value of the C-bit for arithmetic operations; otherwise not affected or set to a specified result.
3	N	Negative condition code bit. Set if the most significant bit of the result is set; otherwise cleared.
2	Z	Zero condition code bit. Set if the result equals zero; otherwise cleared.
1	V	Overflow condition code bit. Set if an arithmetic overflow occurs implying that the result cannot be represented in the operand size; otherwise cleared.
0	C	Carry condition code bit. Set if a carry out of the operand msb occurs for an addition, or if a borrow occurs in a subtraction; otherwise cleared Set to the value of the C bit for arithmetic operations; otherwise not affected.

### 3.2.2 EMAC Register Description

The registers in the EMAC portion of the user programming model, are described in [Chapter 4](#), “Enhanced Multiply-Accumulate Unit (EMAC),” and include the following registers:

- Four 48-bit accumulator registers partitioned as follows:
  - Four 32-bit accumulators (ACC0–ACC3)
  - Eight 8-bit accumulator extension bytes (two per accumulator). These are grouped into two 32-bit values for load and store operations (ACCEXT01 and ACCEXT23).

Accumulators and extension bytes can be loaded, copied, and stored, and results from EMAC arithmetic operations generally affect the entire 48-bit destination.
- Eight 8-bit accumulator extensions (two per accumulator), packaged as two 32-bit values for load and store operations (ACCext01 and ACCext23)
- One 16-bit mask register (MASK)

- One 32-bit status register (MACSR) including four indicator bits signaling product or accumulation overflow (one for each accumulator: PAV0–PAV3)

These registers are shown in [Table 3-2](#).

**Table 3-2. EMAC Register Set**

31:24	23:16	15:8	7:0	Mnemonic
MAC Status Register				MACSR
MAC Accumulator 0				ACC0
MAC Accumulator 1				ACC1
MAC Accumulator 2				ACC2
MAC Accumulator 3				ACC3
Extensions for ACC0 and ACC1				ACCext01
Extensions for ACC2 and ACC3				ACCext23
MAC Mask Register				MASK

### 3.2.3 Supervisor Register Description

Only system control software is intended to use the supervisor programming model to implement restricted operating system functions, I/O control, and memory management. All accesses that affect the control features of ColdFire processors are in the supervisor programming model, which consists of registers available in user mode as well as the following control registers:

- 16-bit status register (SR)
- 32-bit supervisor stack pointer (SSP)
- 32-bit vector base register (VBR)
- 32-bit cache control register (CACR)
- Two 32-bit access control registers (ACR0, ACR1)
- Two 32-bit base address registers (RAMBAR)

**Table 3-3. Supervisor Programming Model**

31:24	23:16	15:8	7:0	Mnemonic
—		Status Register		SR
Supervisor/User A7 Stack Pointer				A7
User/Supervisor A7 Stack Pointer				OTHER_A7
Vector Base Register				VBR
Cache Control Register				CACR
Access Control Register 0				ACR0

**Table 3-3. Supervisor Programming Model**

31:24	23:16	15:8	7:0	Mnemonic
Access Control Register 1				ACR1
RAM Base Address Register				RAMBAR1

The following paragraphs describe the supervisor programming model registers.

### 3.2.3.1 Status Register (SR)

The SR stores the processor status and includes the CCR, the interrupt priority mask, and other control bits. In supervisor mode, software can access the entire SR. In user mode, only the lower 8 bits are accessible (CCR). The control bits indicate the following states for the processor: trace mode (T bit), supervisor or user mode (S bit), and master or interrupt state (M bit). All defined bits in the SR have read/write access when in supervisor mode.

System Byte								Condition Code Register (CCR)								
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Field	T	—	S	M	—	I			—			X	N	Z	V	C
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Address	CPU @ 0x80E															

**Figure 3-4. Status Register (SR)****Table 3-4. SR Field Descriptions**

Bits	Name	Description
15	T	Trace enable. When set, the processor performs a trace exception after every instruction.
14	—	Reserved, should be cleared.
13	S	Supervisor/user state. Denotes whether the processor is in supervisor mode (S = 1) or user mode (S = 0).
12	M	Master/interrupt state. This bit is cleared by an interrupt exception, and can be set by software during execution of the RTE or move to SR instructions.
11	—	Reserved, should be cleared.
10–8	I	Interrupt level mask. Defines the current interrupt level. Interrupt requests are inhibited for all priority levels less than or equal to the current level, except the edge-sensitive level 7 request, which cannot be masked.
7–5	—	Reserved, should be cleared.
4–0	CCR	Refer to <a href="#">Table 3-1</a> .

### 3.2.3.2 Supervisor/User Stack Pointers (A7 and OTHER\_A7)

The MCF5275 architecture supports two independent stack pointer (A7) registers—the supervisor stack pointer (SSP) and the user stack pointer (USP). The hardware implementation of these two programmable-visible 32-bit registers does not identify one as the SSP and the other as the USP. Instead, the hardware uses one 32-bit register as the active A7 and the other as OTHER\_A7. Thus, the register contents are a function of the processor operation mode, as shown in the following:

```
if SR[S] = 1
    then    A7 = Supervisor Stack Pointer
           OTHER_A7 = User Stack Pointer
    else    A7 = User Stack Pointer
           OTHER_A7 = Supervisor Stack Pointer
```

The BDM programming model supports direct reads and writes to A7 and OTHER\_A7. It is the responsibility of the external development system to determine, based on the setting of SR[S], the mapping of A7 and OTHER\_A7 to the two program-visible definitions (SSP and USP). This functionality is enabled by setting the enable user stack pointer bit, CACR[EUSP]. If this bit is cleared, only the stack pointer (A7), defined for previous ColdFire versions, is available. EUSP is zero at reset.

If EUSP is set, the appropriate stack pointer register (SSP or USP) is accessed as a function of the processor's operating mode. To support dual stack pointers, the following two privileged M68000 instructions are added to the ColdFire instruction set architecture to load/store the USP:

```
move.l Ay, USP; move to USP
move.l USP, Ax; move from USP
```

These instructions are described in the *ColdFire Family Programmer's Reference Manual*.

### 3.2.3.3 Vector Base Register (VBR)

The VBR contains the base address of the exception vector table in memory. To access the vector table, the displacement of an exception vector is added to the value in VBR. The lower 20 bits of the VBR are not implemented by ColdFire processors; they are assumed to be zero, forcing the table to be aligned on a 1 MByte boundary.

### 3.2.3.4 Cache Control Register (CACR)

The CACR controls operation of the instruction/data cache memories. It includes bits for enabling, freezing, and invalidating cache contents. It also includes bits for defining the default cache mode and write-protect fields. The CACR is described in [Section 5.2.1.1, “Cache Control Register \(CACR\).”](#)

### 3.2.3.5 Access Control Registers (ACR0, ACR1)

The access control registers, ACR0 and ACR1, define attributes for two user-defined memory regions. These attributes include the definition of cache mode, write protect, and buffer write enables. The ACRs are described in [Section 5.2.1.2, “Access Control Registers \(ACR0, ACR1\).”](#)

### 3.2.3.6 SRAM Base Address Register (RAMBAR)

The RAMBAR register is used to specify the base address of the internal SRAM and indicate the types of references mapped to it. The base address register includes a base address, write-protect bit, address space mask bits, and an enable bit. For more information, refer to [Section 6.2.1, “SRAM Base Address Register \(RAMBAR\)”](#).

## 3.3 Memory Map/Register Definition

[Table 3-5](#) lists register names, the CPU space location, and whether the register is written from the processor using the MOVEC instruction.

**Table 3-5. ColdFire CPU Registers**

Name	CPU Space (Rc)	Written with MOVEC	Register Name
<b>Memory Management Control Registers</b>			
CACR	0x002	Yes	Cache control register
ACR0, ACR1	0x004–0x005	Yes	Access control registers 0 and 1
<b>Processor General-Purpose Registers</b>			
D0–D7	0x(0,1)80–0x(0,1)87	No	Data registers 0-7 (0 = load, 1 = store)
A0–A7	0x(0,1)88–0x(0,1)8F	No	Address registers 0-7 (0 = load, 1 = store) A7 is user stack pointer
<b>Processor Miscellaneous Registers</b>			
OTHER_A7	0x800	No	Other stack pointer
VBR	0x801	Yes	Vector base register
MACSR	0x804	No	MAC status register
MASK	0x805	No	MAC address mask register
ACC0–ACC3	0x806, 0x809, 0x80A, 0x80B	No	MAC accumulators 0-3
ACCext01	0x807	No	MAC accumulator 0, 1 extension bytes
ACCext23	0x808	No	MAC accumulator 2, 3 extension bytes
SR	0x80E	No	Status register
PC	0x80F	Yes	Program counter

**Table 3-5. ColdFire CPU Registers (Continued)**

Name	CPU Space (Rc)	Written with MOVEC	Register Name
<b>Local Memory Registers</b>			
RAMBAR	0xC05	Yes	SRAM base address register

## 3.4 Additions to the Instruction Set Architecture

The original ColdFire instruction set architecture (ISA) was derived from the M68000-family opcodes based on extensive analysis of embedded application code. After the initial ColdFire compilers were created, developers identified ISA additions that would enhance both code density and overall performance. Additionally, as users implemented ColdFire-based designs into a wide range of embedded systems, they identified frequently used instruction sequences that could be improved by the creation of new instructions. This observation was especially prevalent in development environments that made use of substantial amounts of assembly language code.

[Table 3-6](#) summarizes the new instructions added to Revision A+ ISA. For more details see [Section 3.14, “ColdFire Instruction Set Architecture Enhancements.”](#)

**Table 3-6. ISA Revision A+ New Instructions**

Instruction	Description
BITREV	The contents of the destination data register are bit-reversed; that is, new Dx[31] = old Dx[0], new Dx[30] = old Dx[1], ..., new Dx[0] = old Dx[31].
BYTEREV	The contents of the destination data register are byte-reversed; that is, new Dx[31:24] = old Dx[7:0], ..., new Dx[7:0] = old Dx[31:24].
FF1	The data register, Dx, is scanned, beginning from the most-significant bit (Dx[31]) and ending with the least-significant bit (Dx[0]), searching for the first set bit. The data register is then loaded with the offset count from bit 31 where the first set bit appears.
STLDSR	Pushes the contents of the status register onto the stack and then reloads the status register with the immediate data value.

## 3.5 Exception Processing Overview

Exception processing for ColdFire processors is streamlined for performance. The ColdFire processors differ from the M68000 family in that they include:

- A simplified exception vector table
- Reduced relocation capabilities using the vector base register
- A single exception stack frame format
- Use of a single self-aligning system stack

All ColdFire processors use an instruction restart exception model, but certain microarchitectures (V2 and V3) require more software support to recover from certain access errors. See [Section 3.7.1, “Access Error Exception”](#) for details.

Exception processing includes all actions from the detection of the fault condition to the initiation of fetch for the first handler instruction. Exception processing is comprised of four major steps

First, the processor makes an internal copy of the SR and then enters supervisor mode by asserting the S bit and disabling trace mode by negating the T bit. The occurrence of an interrupt exception also forces the M bit to be cleared and the interrupt priority mask to be set to the level of the current interrupt request.

Second, the processor determines the exception vector number. For all faults *except* interrupts, the processor performs this calculation based on the exception type. For interrupts, the processor performs an interrupt-acknowledge (IACK) bus cycle to obtain the vector number from the interrupt controller. The IACK cycle is mapped to a special acknowledge address space with the interrupt level encoded in the address.

Third, the processor saves the current context by creating an exception stack frame on the supervisor system stack. As a result, the exception stack frame is created at a 0-modulo-4 address on the top of the current system stack. Additionally, the processor uses a simplified fixed-length stack frame for all exceptions. The exception type determines whether the program counter placed in the exception stack frame defines the location of the faulting instruction (fault) or the address of the next instruction to be executed (next).

Fourth, the processor calculates the address of the first instruction of the exception handler. By definition, the exception vector table is aligned on a 1 Mbyte boundary. This instruction address is generated by fetching an exception vector from the table located at the address defined in the vector base register. The index into the exception table is calculated as (4 x vector number). Once the exception vector has been fetched, the contents of the vector determine the address of the first instruction of the desired handler. After the instruction fetch for the first opcode of the handler has been initiated, exception processing terminates and normal instruction processing continues in the handler.

All ColdFire processors support a 1024-byte vector table aligned on any 1 Mbyte address boundary (see [Table 3-7](#)). The table contains 256 exception vectors; the first 64 are defined by Freescale and the remaining 192 are user-defined interrupt vectors.

**Table 3-7. Exception Vector Assignments**

Vector Number(s)	Vector Offset (Hex)	Stacked Program Counter	Assignment
0	0x000	—	Initial stack pointer
1	0x004	—	Initial program counter
2	0x008	Fault	Access error



**Table 3-7. Exception Vector Assignments (Continued)**

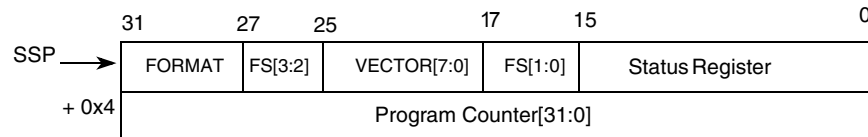
Vector Number(s)	Vector Offset (Hex)	Stacked Program Counter	Assignment
3	0x00C	Fault	Address error
4	0x010	Fault	Illegal instruction
5	0x014	Fault	Divide by zero
6–7	0x018–0x01C	—	Reserved
8	0x020	Fault	Privilege violation
9	0x024	Next	Trace
10	0x028	Fault	Unimplemented line-a opcode
11	0x02C	Fault	Unimplemented line-f opcode
12	0x030	Next	Debug interrupt
13	0x034	—	Reserved
14	0x038	Fault	Format error
15–23	0x03C–0x05C	—	Reserved
24	0x060	Next	Spurious interrupt
25–31	0x064–0x07C	—	Reserved
32–47	0x080–0x0BC	Next	Trap # 0-15 instructions
48–63	0x0C0–0x0FC	—	Reserved
64–255	0x100–0x3FC	Next	User-defined interrupts

“Fault” refers to the PC of the instruction that caused the exception; “Next” refers to the PC of the next instruction that follows the instruction that caused the fault.

All ColdFire processors inhibit interrupt sampling during the first instruction of all exception handlers. This allows any handler to effectively disable interrupts, if necessary, by raising the interrupt mask level contained in the status register. In addition, the V2 core includes a new instruction (STLDSR) that stores the current interrupt mask level and loads a value into the SR. This instruction is specifically intended for use as the first instruction of an interrupt service routine which services multiple interrupt requests with different interrupt levels. For more details see [Section 3.14, “ColdFire Instruction Set Architecture Enhancements.”](#)

## 3.6 Exception Stack Frame Definition

The exception stack frame is shown in [Figure 3-5](#). The first longword of the exception stack frame contains the 16-bit format/vector word (F/V) and the 16-bit status register, and the second longword contains the 32-bit program counter address.

**Figure 3-5. Exception Stack Frame Form**

The 16-bit format/vector word contains 3 unique fields:

- A 4-bit format field at the top of the system stack is always written with a value of 4, 5, 6, or 7 by the processor indicating a two-longword frame format. See [Table 3-8](#).

**Table 3-8. Format Field Encodings**

Original SSP @ Time of Exception, Bits 1:0	SSP @ 1st Instruction of Handler	Format Field
00	Original SSP - 8	4
01	Original SSP - 9	5
10	Original SSP - 10	6
11	Original SSP - 11	7

- There is a 4-bit fault status field, FS[3:0], at the top of the system stack. This field is defined for access and address errors only and written as zeros for all other types of exceptions. See [Table 3-9](#).

**Table 3-9. Fault Status Encodings**

FS[3:0]	Definition
00xx	Reserved
0100	Error on instruction fetch
0101	Reserved
011x	Reserved
1000	Error on operand write
1001	Attempted write to write-protected space
101x	Reserved
1100	Error on operand read
1101	Reserved
111x	Reserved

- The 8-bit vector number, vector[7:0], defines the exception type and is calculated by the processor for all internal faults and represents the value supplied by the interrupt controller in the case of an interrupt. Refer to [Table 3-7](#).

## 3.7 Processor Exceptions

### 3.7.1 Access Error Exception

The exact processor response to an access error depends on the type of memory reference being performed. For an instruction fetch, the processor postpones the error reporting until the faulted reference is needed by an instruction for execution. Therefore, faults that occur during instruction prefetches that are then followed by a change of instruction flow do not generate an exception. When the processor attempts to execute an instruction with a faulted opword and/or extension words, the access error is signaled and the instruction aborted. For this type of exception, the programming model has not been altered by the instruction generating the access error.

If the access error occurs on an operand read, the processor immediately aborts the current instruction's execution and initiates exception processing. In this situation, any address register updates attributable to the auto-addressing modes, (for example,  $(An)^+$ ,  $-(An)$ ), have already been performed, so the programming model contains the updated  $An$  value. In addition, if an access error occurs during the execution of a MOVEM instruction loading from memory, any registers already updated *before* the fault occurs contain the operands from memory.

The V2 ColdFire processor uses an imprecise reporting mechanism for access errors on operand writes. Because the actual write cycle may be decoupled from the processor's issuing of the operation, the signaling of an access error appears to be decoupled from the instruction that generated the write. Accordingly, the PC contained in the exception stack frame merely represents the location in the program when the access error was signaled. All programming model updates associated with the write instruction are completed. The NOP instruction can collect access errors for writes. This instruction delays its execution until all previous operations, including all pending write operations, are complete. If any previous write terminates with an access error, it is guaranteed to be reported on the NOP instruction.

### 3.7.2 Address Error Exception

Any attempted execution transferring control to an odd instruction address (that is, if bit 0 of the target address is set) results in an address error exception.

Any attempted use of a word-sized index register ( $Xn.w$ ) or a scale factor of 8 on an indexed effective addressing mode generates an address error as does an attempted execution of a full-format indexed addressing mode.

### 3.7.3 Illegal Instruction Exception

Any attempted execution of an illegal 16-bit opcode (except for line-A and line-F opcodes) generates an illegal instruction exception (vector 4). Additionally, any attempted execution of any non-MAC line-A and most line-F opcode generates their unique exception types, vector numbers

10 and 11, respectively. The V2 core does not provide illegal instruction detection on the extension words on any instruction, including MOVEC.

### 3.7.4 Divide-By-Zero

Attempting to divide by zero causes an exception (vector 5, offset = 0x014).

### 3.7.5 Privilege Violation

The attempted execution of a supervisor mode instruction while in user mode generates a privilege violation exception. See the *ColdFire Programmer's Reference Manual* for lists of supervisor- and user-mode instructions.

### 3.7.6 Trace Exception

To aid in program development, all ColdFire processors provide an instruction-by-instruction tracing capability. While in trace mode, indicated by the assertion of the T-bit in the status register (SR[15] = 1), the completion of an instruction execution (for all but the STOP instruction) signals a trace exception. This functionality allows a debugger to monitor program execution.

The STOP instruction has the following effects:

1. The instruction before the STOP executes and then generates a trace exception. In the exception stack frame, the PC points to the STOP opcode.
2. When the trace handler is exited, the STOP instruction is executed, loading the SR with the immediate operand from the instruction.
3. The processor then generates a trace exception. The PC in the exception stack frame points to the instruction after the STOP, and the SR reflects the value loaded in the previous step.

If the processor is not in trace mode and executes a STOP instruction where the immediate operand sets SR[T], hardware loads the SR and generates a trace exception. The PC in the exception stack frame points to the instruction after the STOP, and the SR reflects the value loaded in step 2.

Because ColdFire processors do not support any hardware stacking of multiple exceptions, it is the responsibility of the operating system to check for trace mode after processing other exception types. As an example, consider the execution of a TRAP instruction while in trace mode. The processor will initiate the TRAP exception and then pass control to the corresponding handler. If the system requires that a trace exception be processed, it is the responsibility of the TRAP exception handler to check for this condition (SR[15] in the exception stack frame asserted) and pass control to the trace handler before returning from the original exception.

### 3.7.7 Unimplemented Line-A Opcode

A line-A opcode is defined when bits 15-12 of the opword are 0b1010. This exception is generated by the attempted execution of an undefined line-A opcode.

### 3.7.8 Unimplemented Line-F Opcode

A line-F opcode is defined when bits 15-12 of the opword are 0b1111. This exception is generated by attempted execution of an undefined line-F opcode.

### 3.7.9 Debug Interrupt

This special type of program interrupt is discussed in detail in [Chapter 32, “Debug Support.”](#) This exception is generated in response to a hardware breakpoint register trigger. The processor does not generate an IACK cycle but rather calculates the vector number internally (vector number 12).

### 3.7.10 RTE and Format Error Exception

When an RTE instruction is executed, the processor first examines the 4-bit format field to validate the frame type. For a ColdFire core, any attempted RTE execution where the format is not equal to {4,5,6,7} generates a format error. The exception stack frame for the format error is created without disturbing the original RTE frame and the stacked PC pointing to the RTE instruction.

The selection of the format value provides some limited debug support for porting code from M68000 applications. On M68000 family processors, the SR was located at the top of the stack. On those processors, bit 30 of the longword addressed by the system stack pointer is typically zero. Thus, if an RTE is attempted using this “old” format, it generates a format error on a ColdFire processor.

If the format field defines a valid type, the processor: (1) reloads the SR operand, (2) fetches the second longword operand, (3) adjusts the stack pointer by adding the format value to the auto-incremented address after the fetch of the first longword, and then (4) transfers control to the instruction address defined by the second longword operand within the stack frame.

### 3.7.11 TRAP Instruction Exception

The TRAP #n instruction always forces an exception as part of its execution and is useful for implementing system calls.

### 3.7.12 Interrupt Exception

Interrupt exception processing includes interrupt recognition and the fetch of the appropriate vector from the interrupt controller using an IACK cycle. See [Chapter 13, “Interrupt Controller Modules,”](#) for details on the interrupt controller.

### 3.7.13 Fault-on-Fault Halt

If a ColdFire processor encounters any type of fault during the exception processing of another fault, the processor immediately halts execution with the catastrophic “fault-on-fault” condition. A reset is required to force the processor to exit this halted state.

### 3.7.14 Reset Exception

Asserting the reset input signal to the processor causes a reset exception. The reset exception has the highest priority of any exception; it provides for system initialization and recovery from catastrophic failure. Reset also aborts any processing in progress when the reset input is recognized. Processing cannot be recovered.

The reset exception places the processor in the supervisor mode by setting the S-bit and disables tracing by clearing the T bit in the SR. This exception also clears the M-bit and sets the processor’s interrupt priority mask in the SR to the highest level (level 7). Next, the VBR is initialized to zero (0x00000000). The control registers specifying the operation of any memories (e.g., cache and/or RAM modules) connected directly to the processor are disabled.

#### NOTE

Other implementation-specific supervisor registers are also affected. Refer to each of the modules in this user’s manual for details on these registers.

Once the processor is granted the bus, it then performs two longword read bus cycles. The first longword at address 0 is loaded into the stack pointer and the second longword at address 4 is loaded into the program counter. After the initial instruction is fetched from memory, program execution begins at the address in the PC. If an access error or address error occurs before the first instruction is executed, the processor enters the fault-on-fault halted state.

ColdFire processors load hardware configuration information into the D0 and D1 general-purpose registers after system reset. The hardware configuration information is loaded immediately after the reset-in signal is negated. This allows an emulator to read out the contents of these registers via BDM to determine the hardware configuration.

Information loaded into D0 defines the processor hardware configuration as shown in [Figure 3-6](#).

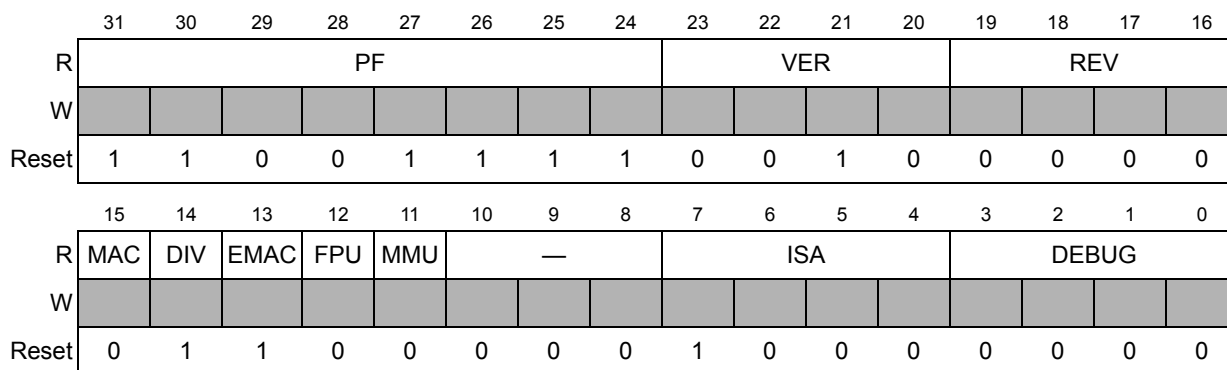


Figure 3-6. D0 Hardware Configuration Info

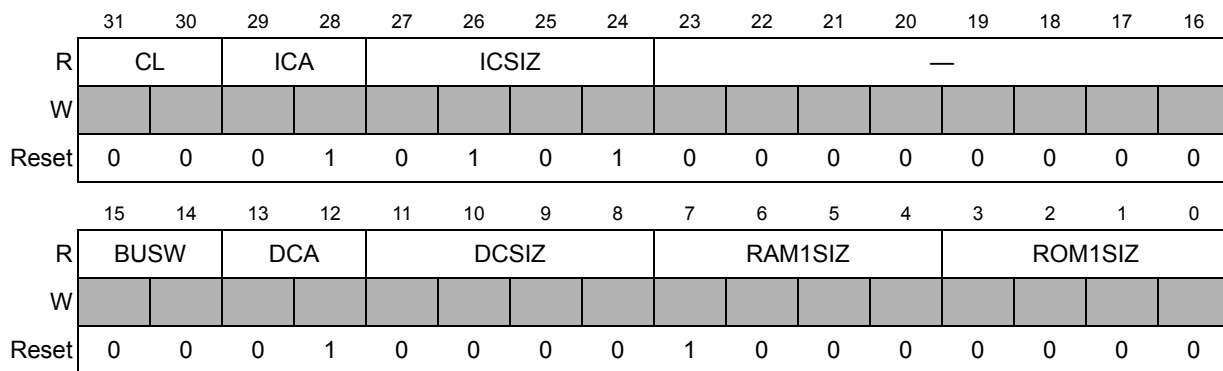
Table 3-10. D0 Hardware Configuration Info Field Description

Bits	Name	Description
31–24	PF	Processor family. This field is fixed to a hex value of 0xCF indicating a ColdFire core is present.
23–20	VER	ColdFire core version number. This field is fixed to a hex value of 0x2 indicating a Version 2 ColdFire core.
19–16	REV	Processor revision number.
15	MAC	MAC execute engine status. 0 MAC execute engine not present in core. (This is the value used for MCF5275 .) 1 MAC execute engine is present in core.
14	DIV	Divide execute engine status. 0 Divide execute engine not present in core. 1 Divide execute engine is present in core. (This is the value used for MCF5275 .)
13	EMAC	EMAC execute engine status. 0 EMAC execute engine not present in core. 1 EMAC execute engine is present in core. (This is the value used for MCF5275)
12	FPU	FPU execute engine status. 0 FPU execute engine not present in core. (This is the value used for MCF5275) 1 FPU execute engine is present in core.
11	MMU	Virtual memory management unit status. 0 MMU execute engine not present in core. (This is the value used for MCF5275) 1 MMU execute engine is present in core.
10–8	—	Reserved.

**Table 3-10. D0 Hardware Configuration Info Field Description (Continued)**

Bits	Name	Description
7–4	ISA	Instruction set architecture (ISA) revision number. 0000 ISA_A 0001 ISA_B 0010 ISA_C 1000 ISA_A+ (ISA_A with the addition of the BYTEREV, BITREV, FF1, and STLDSP instructions. This is the value used for MCF5275.) 0011–1111 Reserved.
3–0	DEBUG	Debug module revision number. 0000 DEBUG_A (This is the value used for MCF5275) 0001 DEBUG_B 0010 DEBUG_C 0011 DEBUG_D 0100 DEBUG_E 0101–1111 Reserved.

Information loaded into D1 defines the local memory hardware configuration as shown in Figure 3-7.

**Figure 3-7. D1 Hardware Configuration Info****Table 3-11. D1 Local Memory Hardware Configuration Information Field Description**

Bits	Name	Description
31–30	CL	Cache line size. This field is fixed to a hex value of 0x0 indicating a 16-byte cache line size.
29–28	ICA	Instruction cache associativity. 00 Four-way. 01 Direct mapped. (This is the value used for MCF5275)
27–24	ICSIZ	Instruction cache size. 0101 8KB instruction cache. (This is the value used for MCF5275) All other values do not apply for MCF5275.
23–16	—	Reserved for MCF5275.
15–14	BUSW	Encoded bus data width. 00 32-bit data bus (only configuration currently in use).



**Table 3-11. D1 Local Memory Hardware Configuration Information Field Description**

Bits	Name	Description
13–12	DCA	Data cache associativity. 00 Four-way. 01 Direct mapped. (This is the value used for MCF5275)
11–8	DCSIZ	Data cache size. 0000 No data cache. (This is the value used for MCF5275) All other values do not apply for MCF5275.
7–4	RAM1SIZ	RAM bank 1 size. 1000 64KB RAM. (This is the value used for MCF5275) All other values do not apply for MCF5275.
3–0	ROM1SIZ	ROM bank 1 size. 0x0–0x3 No ROM. (This is the value used for MCF5275) All other values do not apply for MCF5275.

## 3.8 Instruction Execution Timing

This section presents V2 processor instruction execution times in terms of processor core clock cycles. The number of operand references for each instruction is enclosed in parentheses following the number of processor clock cycles. Each timing entry is presented as C(R/W) where:

- C is the number of processor clock cycles, including all applicable operand fetches and writes, and all internal core cycles required to complete the instruction execution.
- R/W is the number of operand reads (R) and writes (W) required by the instruction. An operation performing a read-modify-write function is denoted as (1/1).

This section includes the assumptions concerning the timing values and the execution time details.

### 3.8.1 Timing Assumptions

For the timing data presented in this section, the following assumptions apply:

1. The operand execution pipeline (OEP) is loaded with the opword and all required extension words at the beginning of each instruction execution. This implies that the OEP does not wait for the instruction fetch pipeline (IFP) to supply opwords and/or extension words.
2. The OEP does not experience any sequence-related pipeline stalls. For V2 ColdFire processors, the most common example of this type of stall involves consecutive store operations, excluding the MOVEM instruction. For all STORE operations (except MOVEM), certain hardware resources within the processor are marked as “busy” for two processor clock cycles after the final DSOC cycle of the store instruction. If a subsequent STORE instruction is encountered within this 2-cycle window, it will be stalled until the resource again becomes available. Thus, the maximum pipeline stall involving consecutive STORE operations is 2 cycles. The MOVEM instruction uses a different set of resources and this stall does not apply.

3. The OEP completes all memory accesses without any stall conditions caused by the memory itself. Thus, the timing details provided in this section assume that an infinite zero-wait state memory is attached to the processor core.
4. All operand data accesses are aligned on the same byte boundary as the operand size: that is, 16 bit operands aligned on 0-modulo-2 addresses and 32 bit operands aligned on 0-modulo-4 addresses.

If the operand alignment fails these guidelines, it is misaligned. The processor core decomposes the misaligned operand reference into a series of aligned accesses as shown in [Table 3-12](#).

**Table 3-12. Misaligned Operand References**

Address[1:0]	Size	Kbus Operations	Additional C(R/W)
X1	Word	Byte, Byte	2(1/0) if read 1(0/1) if write
X1	Long	Byte, Word, Byte	3(2/0) if read 2(0/2) if write
10	Long	Word, Word	2(1/0) if read 1(0/1) if write

### 3.8.2 MOVE Instruction Execution Times

The execution times for the MOVE.{B,W} instructions are shown in [Table 3-13](#), while [Table 3-14](#) provides the timing for MOVE.L.

For all tables in this section, the execution time of any instruction using the PC-relative effective addressing modes is the same for the comparable An-relative mode.

The nomenclature “xxx.wl” refers to both forms of absolute addressing, xxx.w and xxx.l.

**Table 3-13. Move Byte and Word Execution Times**

Source	Destination						
	Rx	(Ax)	(Ax)+	-(Ax)	(d <sub>16</sub> ,Ax)	(d <sub>8</sub> ,Ax,Xi)	(xxx).wl
Dn	1(0/0)	1(0/1)	1(0/1)	1(0/1)	1(0/1)	2(0/1)	1(0/1)
An	1(0/0)	1(0/1)	1(0/1)	1(0/1)	1(0/1)	2(0/1)	1(0/1)
(An)	3(1/0)	3(1/1)	3(1/1)	3(1/1)	3(1/1)	4(1/1)	3(1/1)
(An)+	3(1/0)	3(1/1)	3(1/1)	3(1/1)	3(1/1)	4(1/1)	3(1/1)
-(An)	3(1/0)	3(1/1)	3(1/1)	3(1/1)	3(1/1)	4(1/1)	3(1/1)
(d <sub>16</sub> ,An)	3(1/0)	3(1/1)	3(1/1)	3(1/1)	3(1/1)	—	—
(d <sub>8</sub> ,An,Xi)	4(1/0)	4(1/1)	4(1/1)	4(1/1)	—	—	—
(xxx).w	3(1/0)	3(1/1)	3(1/1)	3(1/1)	—	—	—

**Table 3-13. Move Byte and Word Execution Times (Continued)**

Source	Destination						
	Rx	(Ax)	(Ax)+	-(Ax)	(d <sub>16</sub> ,Ax)	(d <sub>8</sub> ,Ax,Xi)	(xxx).wl
(xxx).l	3(1/0)	3(1/1)	3(1/1)	3(1/1)	—	—	—
(d <sub>16</sub> ,PC)	3(1/0)	3(1/1)	3(1/1)	3(1/1)	3(1/1)	—	—
(d <sub>8</sub> ,PC,Xi)	4(1/0)	4(1/1)	4(1/1)	4(1/1)	—	—	—
#<xxx>	1(0/0)	3(0/1)	3(0/1)	3(0/1)	—	—	—

**Table 3-14. Move Long Execution Times**

Source	Destination						
	Rx	(Ax)	(Ax)+	-(Ax)	(d <sub>16</sub> ,Ax)	(d <sub>8</sub> ,Ax,Xi)	(xxx).wl
Dn	1(0/0)	1(0/1)	1(0/1)	1(0/1)	1(0/1)	2(0/1)	1(0/1)
An	1(0/0)	1(0/1)	1(0/1)	1(0/1)	1(0/1)	2(0/1)	1(0/1)
(An)	2(1/0)	2(1/1)	2(1/1)	2(1/1)	2(1/1)	3(1/1)	2(1/1)
(An)+	2(1/0)	2(1/1)	2(1/1)	2(1/1)	2(1/1)	3(1/1)	2(1/1)
-(An)	2(1/0)	2(1/1)	2(1/1)	2(1/1)	2(1/1)	3(1/1)	2(1/1)
(d <sub>16</sub> ,An)	2(1/0)	2(1/1)	2(1/1)	2(1/1)	2(1/1)	—	—
(d <sub>8</sub> ,An,Xi)	3(1/0)	3(1/1)	3(1/1)	3(1/1)	—	—	—
(xxx).w	2(1/0)	2(1/1)	2(1/1)	2(1/1)	—	—	—
(xxx).l	2(1/0)	2(1/1)	2(1/1)	2(1/1)	—	—	—
(d <sub>16</sub> ,PC)	2(1/0)	2(1/1)	2(1/1)	2(1/1)	2(1/1)	—	—
(d <sub>8</sub> ,PC,Xi)	3(1/0)	3(1/1)	3(1/1)	3(1/1)	—	—	—
#<xxx>	1(0/0)	2(0/1)	2(0/1)	2(0/1)	—	—	—

## 3.9 Standard One Operand Instruction Execution Times

**Table 3-15. One Operand Instruction Execution Times**

Opcode	<EA>	Effective Address							
		Rn	(An)	(An)+	-(An)	(d <sub>16</sub> ,An)	(d <sub>8</sub> ,An,Xn*SF)	xxx.wl	#xxx
bitrev	Dx	1(0/0)	—	—	—	—	—	—	—
bytere	Dx	1(0/0)	—	—	—	—	—	—	—
clr.b	<ea>	1(0/0)	1(0/1)	1(0/1)	1(0/1)	1(0/1)	2(0/1)	1(0/1)	—
clr.w	<ea>	1(0/0)	1(0/1)	1(0/1)	1(0/1)	1(0/1)	2(0/1)	1(0/1)	—
clr.l	<ea>	1(0/0)	1(0/1)	1(0/1)	1(0/1)	1(0/1)	2(0/1)	1(0/1)	—

**Table 3-15. One Operand Instruction Execution Times (Continued)**

Opcode	<EA>	Effective Address							
		Rn	(An)	(An)+	-(An)	(d16,An)	(d8,An,Xn*SF)	xxx.wl	#xxx
ext.w	Dx	1(0/0)	—	—	—	—	—	—	—
ext.l	Dx	1(0/0)	—	—	—	—	—	—	—
extb.l	Dx	1(0/0)	—	—	—	—	—	—	—
ff1	Dx	1(0/0)	—	—	—	—	—	—	—
neg.l	Dx	1(0/0)	—	—	—	—	—	—	—
negx.l	Dx	1(0/0)	—	—	—	—	—	—	—
not.l	Dx	1(0/0)	—	—	—	—	—	—	—
scc	Dx	1(0/0)	—	—	—	—	—	—	—
stldsr	#imm	—	—	—	—	—	—	—	5(0/1)
swap	Dx	1(0/0)	—	—	—	—	—	—	—
tst.b	<ea>	1(0/0)	3(1/0)	3(1/0)	3(1/0)	3(1/0)	4(1/0)	3(1/0)	1(0/0)
tst.w	<ea>	1(0/0)	3(1/0)	3(1/0)	3(1/0)	3(1/0)	4(1/0)	3(1/0)	1(0/0)
tst.l	<ea>	1(0/0)	2(1/0)	2(1/0)	2(1/0)	2(1/0)	3(1/0)	2(1/0)	1(0/0)

## 3.10 Standard Two Operand Instruction Execution Times

**Table 3-16. Two Operand Instruction Execution Times**

Opcode	<EA>	Effective Address							
		Rn	(An)	(An)+	-(An)	(d16,An) (d16,PC)	(d8,An,Xn*SF) (d8,PC,Xn*SF)	xxx.wl	#xxx
add.l	<ea>,Rx	1(0/0)	3(1/0)	3(1/0)	3(1/0)	3(1/0)	4(1/0)	3(1/0)	1(0/0)
add.l	Dy,<ea>	—	3(1/1)	3(1/1)	3(1/1)	3(1/1)	4(1/1)	3(1/1)	—
addi.l	#imm,Dx	1(0/0)	—	—	—	—	—	—	—
addq.l	#imm,<ea>	1(0/0)	3(1/1)	3(1/1)	3(1/1)	3(1/1)	4(1/1)	3(1/1)	—
addx.l	Dy,Dx	1(0/0)	—	—	—	—	—	—	—
and.l	<ea>,Rx	1(0/0)	3(1/0)	3(1/0)	3(1/0)	3(1/0)	4(1/0)	3(1/0)	1(0/0)
and.l	Dy,<ea>	—	3(1/1)	3(1/1)	3(1/1)	3(1/1)	4(1/1)	3(1/1)	—
andi.l	#imm,Dx	1(0/0)	—	—	—	—	—	—	—
asl.l	<ea>,Dx	1(0/0)	—	—	—	—	—	—	1(0/0)
asr.l	<ea>,Dx	1(0/0)	—	—	—	—	—	—	1(0/0)
bchg	Dy,<ea>	2(0/0)	4(1/1)	4(1/1)	4(1/1)	4(1/1)	5(1/1)	4(1/1)	—
bchg	#imm,<ea>	2(0/0)	4(1/1)	4(1/1)	4(1/1)	4(1/1)	—	—	—
bclr	Dy,<ea>	2(0/0)	4(1/1)	4(1/1)	4(1/1)	4(1/1)	5(1/1)	4(1/1)	—
bclr	#imm,<ea>	2(0/0)	4(1/1)	4(1/1)	4(1/1)	4(1/1)	—	—	—

**Table 3-16. Two Operand Instruction Execution Times (Continued)**

Opcode	<EA>	Effective Address							
		Rn	(An)	(An)+	-(An)	(d16,An) (d16,PC)	(d8,An,Xn*SF) (d8,PC,Xn*SF)	xxx.wl	#xxx
bset	Dy,<ea>	2(0/0)	4(1/1)	4(1/1)	4(1/1)	4(1/1)	5(1/1)	4(1/1)	—
bset	#imm,<ea>	2(0/0)	4(1/1)	4(1/1)	4(1/1)	4(1/1)	—	—	—
btst	Dy,<ea>	2(0/0)	3(1/1)	3(1/1)	3(1/1)	3(1/1)	4(1/1)	3(1/1)	—
btst	#imm,<ea>	1(0/0)	3(1/1)	3(1/1)	3(1/1)	3(1/1)	—	—	1(0/0)
cmp.l	<ea>,Rx	1(0/0)	3(1/0)	3(1/0)	3(1/0)	3(1/0)	4(1/0)	3(1/0)	1(0/0)
cmpi.l	#imm,Dx	1(0/0)	—	—	—	—	—	—	—
divs.w <sup>1</sup>	<ea>,Dx	20(0/0)	23(1/0)	23(1/0)	23(1/0)	23(1/0)	24(1/0)	23(1/0)	20(0/0)
divu.w <sup>1</sup>	<ea>,Dx	20(0/0)	23(1/0)	23(1/0)	23(1/0)	23(1/0)	24(1/0)	23(1/0)	20(0/0)
divs.l <sup>1</sup>	<ea>,Dx	≤35(0/0)	≤38(1/0)	≤38(1/0)	≤38(1/0)	≤38(1/0)	—	—	—
divu.l <sup>1</sup>	<ea>,Dx	≤35(0/0)	≤38(1/0)	≤38(1/0)	≤38(1/0)	≤38(1/0)	—	—	—
eor.l	Dy,<ea>	1(0/0)	3(1/1)	3(1/1)	3(1/1)	3(1/1)	4(1/1)	3(1/1)	—
eori.l	#imm,Dx	1(0/0)	—	—	—	—	—	—	—
lea	<ea>,Ax	—	1(0/0)	—	—	1(0/0)	2(0/0)	1(0/0)	—
lsl.l	<ea>,Dx	1(0/0)	—	—	—	—	—	—	1(0/0)
lsr.l	<ea>,Dx	1(0/0)	—	—	—	—	—	—	1(0/0)
moveq	#imm,Dx	—	—	—	—	—	—	—	1(0/0)
muls.w	<ea>y, Dx	4(0/0)	6(1/0)	6(1/0)	6(1/0)	6(1/0)	7(1/0)	6(1/0)	4(1/0)
mulu.w	<ea>y, Dx	4(0/0)	6(1/0)	6(1/0)	6(1/0)	6(1/0)	7(1/0)	6(1/0)	4(1/0)
muls.l	<ea>y, Dx	4(0/0)	6(1/0)	6(1/0)	6(1/0)	6(1/0)	—	—	—
mulu.l	<ea>y, Dx	4(0/0)	6(1/0)	6(1/0)	6(1/0)	6(1/0)	—	—	—
or.l	<ea>,Rx	1(0/0)	3(1/0)	3(1/0)	3(1/0)	3(1/0)	4(1/0)	3(1/0)	1(0/0)
or.l	Dy,<ea>	—	3(1/1)	3(1/1)	3(1/1)	3(1/1)	4(1/1)	3(1/1)	—
ori.l	#imm,Dx	1(0/0)	—	—	—	—	—	—	—
rems.l <sup>1</sup>	<ea>,Dx	≤35(0/0)	≤38(1/0)	≤38(1/0)	≤38(1/0)	≤38(1/0)	—	—	—
remu.l <sup>1</sup>	<ea>,Dx	≤35(0/0)	≤38(1/0)	≤38(1/0)	≤38(1/0)	≤38(1/0)	—	—	—
sub.l	<ea>,Rx	1(0/0)	3(1/0)	3(1/0)	3(1/0)	3(1/0)	4(1/0)	3(1/0)	1(0/0)
sub.l	Dy,<ea>	—	3(1/1)	3(1/1)	3(1/1)	3(1/1)	4(1/1)	3(1/1)	—
subi.l	#imm,Dx	1(0/0)	—	—	—	—	—	—	—
subq.l	#imm,<ea>	1(0/0)	3(1/1)	3(1/1)	3(1/1)	3(1/1)	4(1/1)	3(1/1)	—
subx.l	Dy,Dx	1(0/0)	—	—	—	—	—	—	—

<sup>1</sup> For divide and remainder instructions the times listed represent the worst-case timing. Depending on the operand values, the actual execution time may be less.

## 3.11 Miscellaneous Instruction Execution Times

Table 3-17. Miscellaneous Instruction Execution Times

Opcode	<EA>	Effective Address							
		Rn	(An)	(An)+	-(An)	(d16,An)	(d8,An,Xn*SF)	xxx.wl	#xxx
link.w	Ay,#imm	2(0/1)	—	—	—	—	—	—	—
move.w	CCR,Dx	1(0/0)	—	—	—	—	—	—	—
move.w	<ea>,CCR	1(0/0)	—	—	—	—	—	—	1(0/0)
move.w	SR,Dx	1(0/0)	—	—	—	—	—	—	—
move.w	<ea>,SR	7(0/0)	—	—	—	—	—	—	7(0/0) <sup>2</sup>
movec	Ry,Rc	9(0/1)	—	—	—	—	—	—	—
movem.l	<ea>,&list	—	1+n(n/0)	—	—	1+n(n/0)	—	—	—
movem.l	&list,<ea>	—	1+n(0/n)	—	—	1+n(0/n)	—	—	—
nop		3(0/0)	—	—	—	—	—	—	—
pea	<ea>	—	2(0/1)	—	—	2(0/1) <sup>4</sup>	3(0/1) <sup>5</sup>	2(0/1)	—
pulse		1(0/0)	—	—	—	—	—	—	—
stop	#imm	—	—	—	—	—	—	—	3(0/0) <sup>3</sup>
trap	#imm	—	—	—	—	—	—	—	15(1/2)
trapf		1(0/0)	—	—	—	—	—	—	—
trapf.w		1(0/0)	—	—	—	—	—	—	—
trapf.l		1(0/0)	—	—	—	—	—	—	—
unlk	Ax	2(1/0)	—	—	—	—	—	—	—
wddata	<ea>	—	3(1/0)	3(1/0)	3(1/0)	3(1/0)	4(1/0)	3(1/0)	3(1/0)
wdebug	<ea>	—	5(2/0)	—	—	5(2/0)	—	—	—

<sup>1</sup>n is the number of registers moved by the MOVEM opcode.

<sup>2</sup>If a MOVE.W #imm,SR instruction is executed and imm[13] = 1, the execution time is 1(0/0).

<sup>3</sup>The execution time for STOP is the time required until the processor begins sampling continuously for interrupts.

<sup>4</sup>PEA execution times are the same for (d16,PC).

<sup>5</sup>PEA execution times are the same for (d8,PC,Xn\*SF).

## 3.12 EMAC Instruction Execution Times

Table 3-18. EMAC Instruction Execution Times

Opcode	<EA>	Effective Address							
		Rn	(An)	(An)+	-(An)	(d16,An)	(d8,An,Xn*SF)	xxx.wl	#xxx
muls.w	<ea>y, Dx	4(0/0)	6(1/0)	6(1/0)	6(1/0)	6(1/0)	7(1/0)	6(1/0)	4(1/0)
mulu.w	<ea>y, Dx	4(0/0)	6(1/0)	6(1/0)	6(1/0)	6(1/0)	7(1/0)	6(1/0)	4(1/0)

**Table 3-18. EMAC Instruction Execution Times (Continued)**

Opcode	<EA>	Effective Address							
		Rn	(An)	(An)+	-(An)	(d16,An)	(d8,An,Xn*SF)	xxx.wl	#xxx
muls.l	<ea>y, Dx	4(0/0)	6(1/0)	6(1/0)	6(1/0)	6(1/0)	—	—	—
mulu.l	<ea>y, Dx	4(0/0)	6(1/0)	6(1/0)	6(1/0)	6(1/0)	—	—	—
mac.w	Ry, Rx, Raccx	1(0/0)	—	—	—	—	—	—	—
mac.l	Ry, Rx, Raccx	1(0/0)	—	—	—	—	—	—	—
msac.w	Ry, Rx, Raccx	1(0/0)	—	—	—	—	—	—	—
msac.l	Ry, Rx, Raccx	1(0/0)	—	—	—	—	—	—	—
mac.w	Ry, Rx, <ea>, Rw, Raccx	—	2(1/0)	2(1/0)	2(1/0)	2(1/0) <sup>1</sup>	—	—	—
mac.l	Ry, Rx, <ea>, Rw, Raccx	—	2(1/0)	2(1/0)	2(1/0)	2(1/0) <sup>1</sup>	—	—	—
msac.w	Ry, Rx, <ea>, Rw	—	2(1/0)	2(1/0)	2(1/0)	2(1/0) <sup>1</sup>	—	—	—
msac.l	Ry, Rx, <ea>, Rw, Raccx	—	2(1/0)	2(1/0)	2(1/0)	2(1/0) <sup>1</sup>	—	—	—
mov.l	<ea>y, Raccx	1(0/0)	—	—	—	—	—	—	1(0/0)
mov.l	Raccy, Raccx	1(0/0)	—	—	—	—	—	—	—
mov.l	<ea>y, MACSR	5(0/0)	—	—	—	—	—	—	5(0/0)
mov.l	<ea>y, Rmask	4(0/0)	—	—	—	—	—	—	4(0/0)
mov.l	<ea>y, Raccext01	1(0/0)	—	—	—	—	—	—	1(0/0)
mov.l	<ea>y, Raccext23	1(0/0)	—	—	—	—	—	—	1(0/0)
mov.l	Raccx, <ea>x	1(0/0) <sup>2</sup>	—	—	—	—	—	—	—
mov.l	MACSR, <ea>x	1(0/0)	—	—	—	—	—	—	—
mov.l	Rmask, <ea>x	1(0/0)	—	—	—	—	—	—	—
mov.l	Raccext01, <ea>x	1(0/0)	—	—	—	—	—	—	—
mov.l	Raccext23, <ea>x	1(0/0)	—	—	—	—	—	—	—

<sup>1</sup> Effective address of (d16,PC) not supported

<sup>2</sup> Storing an accumulator requires one additional processor clock cycle when saturation is enabled, or fractional rounding is performed (MACSR[7:4] = 1---, -11-, --11)

**NOTE**

The execution times for moving the contents of the Racc, Raccext[01,23], MACSR, or Rmask into a destination location <ea>x shown in this table represent the *best-case scenario* when the store instruction is executed and there are *no* load or M{S}AC instructions in the EMAC execution pipeline. In general, these store operations require only a single cycle for execution, but if preceded *immediately* by a load, MAC, or MSAC instruction, the depth of the EMAC pipeline is exposed and the execution time is four cycles.

### 3.13 Branch Instruction Execution Times

**Table 3-19. General Branch Instruction Execution Times**

Opcode	<EA>	Effective Address							
		Rn	(An)	(An)+	-(An)	(d16,An) (d16,PC)	(d8,An,Xi*SF) (d8,PC,Xi*SF)	xxx.wl	#xxx
bsr		—	—	—	—	3(0/1)	—	—	—
jmp	<ea>	—	3(0/0)	—	—	3(0/0)	4(0/0)	3(0/0)	—
jsr	<ea>	—	3(0/1)	—	—	3(0/1)	4(0/1)	3(0/1)	—
rte		—	—	10(2/0)	—	—	—	—	—
rts		—	—	5(1/0)	—	—	—	—	—

**Table 3-20. BRA, Bcc Instruction Execution Times**

Opcode	Forward Taken	Forward Not Taken	Backward Taken	Backward Not Taken
bra	2(0/0)	—	2(0/0)	—
bcc	3(0/0)	1(0/0)	2(0/0)	3(0/0)

### 3.14 ColdFire Instruction Set Architecture Enhancements

This section describes the new opcodes implemented as part of the Revision A+ enhancements to the basic ColdFire ISA.



# BITREV

## Bit Reverse Register (Supported Starting with ISA A+)

# BITREV

**Operation:** Bit Reversed Dx → Dx

**Assembler Syntax:** BITREV.L Dx

**Attributes:** Size = longword

Instruction Format:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	0	0	0	0	0	0	1	1	0	0	0	Register, Dx		

The contents of the destination data register are bit-reversed; that is, new Dx[31] = old Dx[0], new Dx[30] = old Dx[1], ..., new Dx[0] = old Dx[31].

**Condition Codes:** Not affected

### Instruction Field:

- Register field—Specifies the destination data register, Dx.

BITREV	V2, V3 Core (ISA_A)	V4 Core (ISA_B)	V2 Core (ISA_A+)
Opcode present	No	No	Yes

# BYTEREV

## Byte Reverse Register (Supported Starting with ISA A+)

# BYTEREV

**Operation:** Byte Reversed Dx → Dx

**Assembler Syntax:** BYTEREV.L Dx

**Attributes:** Size = longword

Instruction Format:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	0	0	0	0	1	0	1	1	0	0	0	Register, Dx		

The contents of the destination data register are byte-reversed as defined below:

new Dx[31:24]	= old Dx[7:0]
new Dx[23:16]	= old Dx[15:8]
new Dx[15:8]	= old Dx[23:16]
new Dx[7:0]	= old Dx[31:24]

**Condition Codes:** Not affected

**Instruction Field:**

- Register field—Specifies the destination data register, Dx.

BYTEREV	V2, V3 Core (ISA_A)	V4 Core (ISA_B)	V2 Core (ISA_A+)
Opcode present	No	No	Yes

**FF1****Find First One in Register**

(Supported Starting with ISA A+)

**FF1****Operation:** Bit Offset of the First Logical One in Register → Destination**Assembler Syntax:** FF1.L Dx**Attributes:** Size = longword

Instruction Format:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	0	0	0	1	0	0	1	1	0	0	0	Destination Register, Dx		

The data register, Dx, is scanned, beginning from the most-significant bit (Dx[31]) and ending with the least-significant bit (Dx[0]), searching for the first set bit. The data register is then loaded with the offset count from bit 31 where the first set bit appears, as shown below. If the source data is zero, then an offset of 32 is returned.

Old Dx[31:0]	New Dx[31:0]
0b1---- . . . ----	0x0000 0000
0b01--- . . . ----	0x0000 0001
0b001-- . . . ----	0x0000 0002
...	...
0b00000 . . . 0010	0x0000 001E
0b00000 . . . 0001	0x0000 001F
0b00000 . . . 0000	0x0000 0020

Condition Codes:	X	N	Z	V	C
	—	*	*	0	0

X Not affected  
 N Set if the msb of the source operand is set; cleared otherwise  
 Z Set if the source operand is zero; cleared otherwise  
 V Always cleared  
 C Always cleared

**Instruction Field:**

- Destination Register field—Specifies the destination data register, Dx.

FF1	V2, V3 Core (ISA_A)	V4 Core (ISA_B)	V2 Core (ISA_A+)
Opcode present	No	No	Yes

# STRLDSR

## Store/Load Status Register

# STRLDSR

(Supported Starting with ISA A+)

**Operation:** If Supervisor State  
 Then SP - 4 → SP; zero-filled SR → (SP); immediate data → SR  
 Else TRAP

**Assembler Syntax:** STRLDSR #<data>

**Attributes:** Size = word

Instruction Format:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	1	0	0	0	0	0	0	1	1	1	0	0	1	1	1
	0	1	0	0	0	1	1	0	1	1	1	1	1	1	0	0
Immediate Data																

**Description:** Pushes the contents of the Status Register onto the stack and then reloads the Status Register with the immediate data value. This instruction is intended for use as the first instruction of an interrupt service routine shared across multiple interrupt request levels. It allows the level of the just-taken interrupt request to be stored in memory (using the SR[IML] field), and then masks interrupts by loading the SR[IML] field with 0x7 (if desired). If execution is attempted with bit 13 of the immediate data cleared (attempting to place the processor in user mode), a privilege violation exception is generated. The opcode for STRLDSR is 0x40E7 46FC.

Condition Codes:	X	N	Z	V	C	
	*	*	*	*	*	

X Set to the value of bit 4 of the immediate operand

N Set to the value of bit 3 of the immediate operand

Z Set to the value of bit 2 of the immediate operand

V Set to the value of bit 1 of the immediate operand

C Set to the value of bit 0 of the immediate operand

STRLDSR	V2, V3 Core (ISA_A)	V4 Core (ISA_B)	V2 Core (ISA_A+)
Opcode present	No	No	Yes

# Chapter 4

## Enhanced Multiply-Accumulate Unit (EMAC)

This chapter describes the functionality, microarchitecture, and performance of the enhanced multiply-accumulate (EMAC) unit in the ColdFire family of processors.

### 4.1 Multiply-Accumulate Unit

The MAC design provides a set of DSP operations which can be used to improve the performance of embedded code while supporting the integer multiply instructions of the baseline ColdFire architecture.

The MAC provides functionality in three related areas:

- Signed and unsigned integer multiplies
- Multiply-accumulate operations supporting signed and unsigned integer operands as well as signed, fixed-point, fractional operands
- Miscellaneous register operations

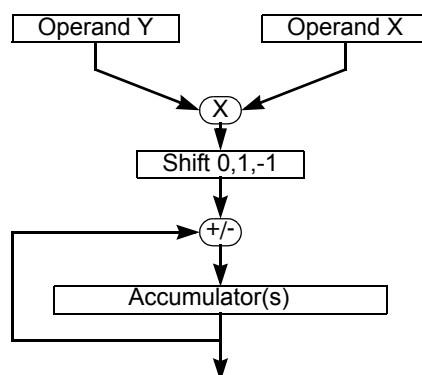
The ColdFire family supports two MAC implementations with different performance levels and capabilities. The original MAC uses a three-stage execution pipeline optimized for 16-bit operands and featuring a 16x16 multiply array with a single 32-bit accumulator. The EMAC features a four-stage pipeline optimized for 32-bit operands, with a fully pipelined  $32 \times 32$  multiply array and four 48-bit accumulators.

The first ColdFire MAC supported signed and unsigned integer operands and was optimized for 16x16 operations, such as those found in a variety of applications including servo control and image compression. As ColdFire-based systems proliferated, the desire for more precision on input operands increased. The result was an improved ColdFire MAC with user-programmable control to optionally enable use of fractional input operands.

EMAC improvements target three primary areas:

- Improved performance of  $32 \times 32$  multiply operations.
- Addition of three more accumulators to minimize MAC pipeline stalls caused by exchanges between the accumulator and the pipeline's general-purpose registers.
- A 48-bit accumulation data path to allow the use of a 40-bit product plus the addition of 8 extension bits to increase the dynamic number range when implementing signal processing algorithms.

The three areas of functionality are addressed in detail in following sections. The logic required to support this functionality is contained in a MAC module, as shown in [Figure 4-1](#).



**Figure 4-1. Multiply-Accumulate Functionality Diagram**

## 4.2 Introduction to the MAC

The MAC is an extension of the basic multiplier found in most microprocessors. It is typically implemented in hardware within an architecture and supports rapid execution of signal processing algorithms in fewer cycles than comparable non-MAC architectures. For example, small digital filters can tolerate some variance in an algorithm's execution time, but larger, more complicated algorithms such as orthogonal transforms may have more demanding speed requirements beyond the scope of any processor architecture and may require full DSP implementation.

To strike a balance between speed, size, and functionality, the ColdFire MAC is optimized for a small set of operations that involve multiplication and cumulative additions. Specifically, the multiplier array is optimized for single-cycle pipelined operations with a possible accumulation after product generation. This functionality is common in many signal processing applications. The ColdFire core architecture also has been modified to allow an operand to be fetched in parallel with a multiply, increasing overall performance for certain DSP operations.

Consider a typical filtering operation where the filter is defined as in [Figure 4-2](#).

$$y(i) = \sum_{k=1}^{N-1} a(k)y(i-k) + \sum_{k=0}^{N-1} b(k)x(i-k)$$

**Figure 4-2. Infinite Impulse Response (IIR) Filter**

Here, the output  $y(i)$  is determined by past output values and past input values. This is the general form of an infinite impulse response (IIR) filter. A finite impulse response (FIR) filter can be obtained by setting coefficients  $a(k)$  to zero. In either case, the operations involved in computing such a filter are multiplies and product summing. To show this point, reduce the above equation to a simple, four-tap FIR filter, shown in [Figure 4-3](#), in which the accumulated sum is a sum of past data values and coefficients.

$$y(i) = \sum_{k=0}^3 b(k)x(i-k) = b(0)x(i) + b(1)x(i-1) + b(2)x(i-2) + b(3)x(i-3)$$

**Figure 4-3. Four-Tap FIR Filter**

## 4.3 General Operation

The MAC speeds execution of ColdFire integer multiply instructions (MULS and MULU) and provides additional functionality for multiply-accumulate operations. By executing MULS and MULU in the MAC, execution times are minimized and deterministic compared to the 2-bit/cycle algorithm with early termination that the OEP normally uses if no MAC hardware is present.

The added MAC instructions to the ColdFire ISA provide for the multiplication of two numbers, followed by the addition or subtraction of the product to or from the value in an accumulator. Optionally, the product may be shifted left or right by 1 bit before addition or subtraction. Hardware support for saturation arithmetic can be enabled to minimize software overhead when dealing with potential overflow conditions. Multiply-accumulate operations support 16- or 32-bit input operands of the following formats:

- Signed integers
- Unsigned integers
- Signed, fixed-point, fractional numbers

The EMAC is optimized for single-cycle, pipelined  $32 \times 32$  multiplications. For word- and longword-sized integer input operands, the low-order 40 bits of the product are formed and used with the destination accumulator. For fractional operands, the entire 64-bit product is calculated and either truncated or rounded to the most-significant 40-bit result using the round-to-nearest (even) method before it is combined with the destination accumulator.

For all operations, the resulting 40-bit product is extended to a 48-bit value (using sign-extension for signed integer and fractional operands, zero-fill for unsigned integer operands) before being combined with the 48-bit destination accumulator.

Figure 4-4 and Figure 4-5 show relative alignment of input operands, the full 64-bit product, the resulting 40-bit product used for accumulation, and 48-bit accumulator formats.

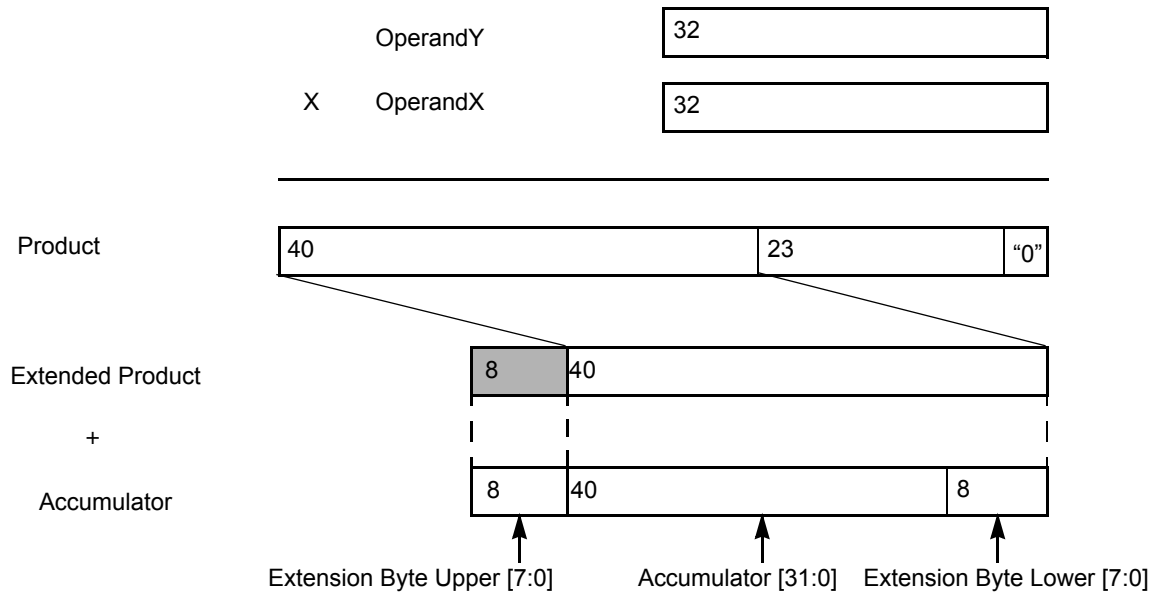


Figure 4-4. Fractional Alignment

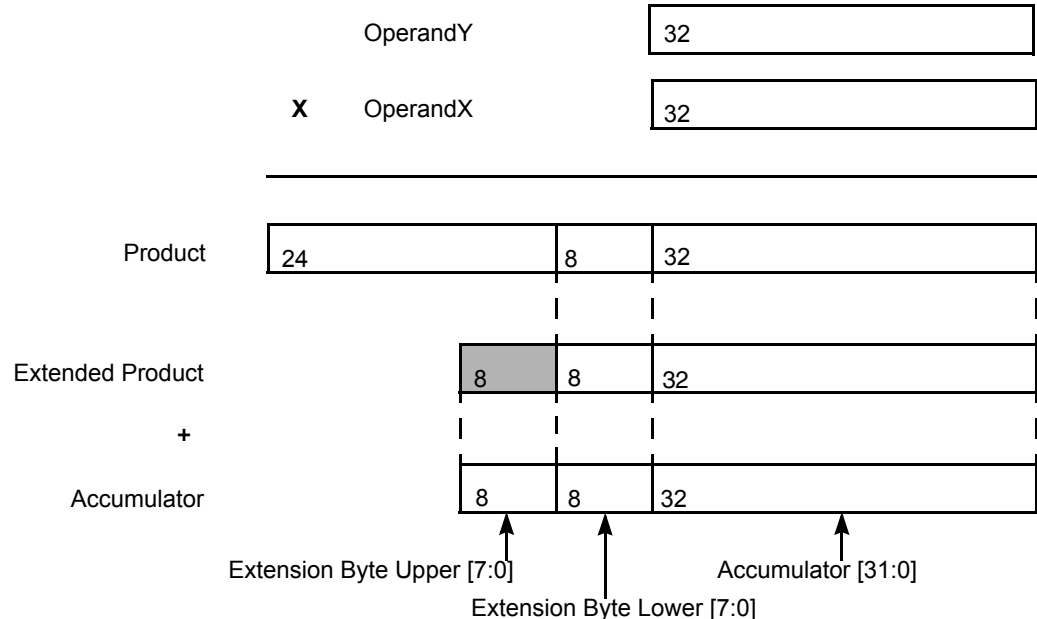


Figure 4-5. Signed and Unsigned Integer Alignment

Thus, the 48-bit accumulator definition is a function of the EMAC operating mode. Given that each 48-bit accumulator is the concatenation of 16-bit accumulator extension register (*ACCextn*) contents and 32-bit *ACCn* contents, the specific definitions are as follows:

```
if MACSR[6:5] == 00/* signed integer mode */
    Complete Accumulator[47:0] = {ACCextn[15:0], ACCn[31:0]}
```



```

if MACSR[6:5] == -1/* signed fractional mode */
    Complete Accumulator [47:0] = {ACCextn[15:8], ACCn[31:0], ACCextn[7:0]}
if MACSR[6:5] == 10/* unsigned integer mode */
    Complete Accumulator[47:0] = {ACCextn[15:0], ACCn[31:0]}

```

The four accumulators are represented as an array,  $ACCn$ , where  $n$  selects the register.

Although the multiplier array is implemented in a four-stage pipeline, all arithmetic MAC instructions have an effective issue rate of 1 cycle, regardless of input operand size or type.

All arithmetic operations use register-based input operands, and summed values are stored internally in an accumulator. Thus, an additional move instruction is needed to store data in a general-purpose register. One new feature found in EMAC instructions is the ability to choose the upper or lower word of a register as a 16-bit input operand. This is useful in filtering operations if one data register is loaded with the input data and another is loaded with the coefficient. Two 16-bit multiply accumulates can be performed without fetching additional operands between instructions by alternating the word choice during the calculations.

The EMAC has four accumulator registers versus the MAC's single accumulator. The additional registers improve the performance of some algorithms by minimizing pipeline stalls needed to store an accumulator value back to general-purpose registers. Many algorithms require multiple calculations on a given data set. By applying different accumulators to these calculations, it is often possible to store one accumulator without any stalls while performing operations involving a different destination accumulator.

The need to move large amounts of data presents an obstacle to obtaining high throughput rates in DSP engines. New and existing ColdFire instructions can accommodate these requirements. A MOVEM instruction can move large blocks of data efficiently by generating line-sized burst references. The ability to simultaneously load an operand from memory into a register and execute a MAC instruction makes some DSP operations such as filtering and convolution more manageable.

The programming model includes a 16-bit mask register (MASK), which can optionally be used to generate an operand address during MAC + MOVE instructions. The application of this register with auto-increment addressing mode supports efficient implementation of circular data queues for memory operands.

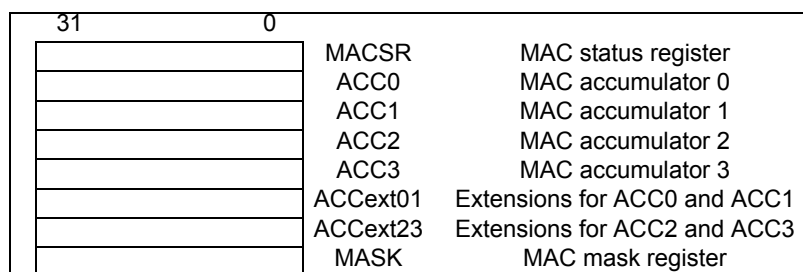
The additional MAC status register (MACSR) contains a 4-bit operational mode field and condition flags. Operational mode bits control whether operands are signed or unsigned and whether they are treated as integers or fractions. These bits also control the overflow/saturation mode and the way in which rounding is performed. Negative, zero, and multiple overflow condition flags are also provided.

## 4.4 Memory Map/Register Definition

The EMAC provides the following program-visible registers:

- Four 32-bit accumulators ( $ACC_n = ACC0, ACC1, ACC2, \text{ and } ACC3$ )
- Eight 8-bit accumulator extensions (two per accumulator), packaged as two 32-bit values for load and store operations ( $ACCext01$  and  $ACCext23$ )
- One 16-bit mask register ( $MASK$ )
- One 32-bit MAC status register ( $MACSR$ ) including four indicator bits signaling product or accumulation overflow (one for each accumulator:  $PAV0$ – $PAV3$ )

These registers are shown in [Figure 4-6](#).



**Figure 4-6. EMAC Register Set**

### 4.4.1 MAC Status Register (MACSR)

MACSR functionality is organized as follows:

- $MACSR[11-8]$  contains one product/accumulation overflow flag per accumulator.
- $MACSR[7-4]$  defines the operating configuration of the MAC unit.
- $MACSR[3-0]$  contains indicator flags from the last MAC instruction execution.

	31	12	11-8	7	6	5	4	3	2	1	0
Field											
Reset	0000_0000_0000_0000_0000_0000_0000_0000										
R/W	R/W										

**Figure 4-7. MAC Status Register (MACSR)**

[Table 4-1](#) describes MACSR fields.

**Table 4-1. MACSR Field Descriptions**

Bits	Name	Description
31–12	—	Reserved, should be cleared.
11–8	PAVx	Product/accumulation overflow flags. Contains four flags, one per accumulator, that indicate if past MAC or MSAC instructions generated an overflow during product calculation or the 48-bit accumulation. When a MAC or MSAC instruction is executed, the PAVx flag associated with the destination accumulator is used to form the general overflow flag, MACSR[V]. Once set, each flag remains set until V is cleared by a MOV.L , MACSR instruction or the accumulator is loaded directly.
7–4	<b>Operational Mode Fields</b>	
7	OMC	Overflow/saturation mode. Used to enable or disable saturation mode on overflow. If set, the accumulator is set to the appropriate constant on any operation which overflows the accumulator. Once saturated, the accumulator remains unaffected by any other MAC or MSAC instructions until either the overflow bit is cleared or the accumulator is directly loaded.
6	S/U	<p>Signed/unsigned operations.</p> <p><b>In integer mode:</b> S/U determines whether operations performed are signed or unsigned. It also determines the accumulator value during saturation, if enabled.</p> <p>0 Signed numbers. On overflow, if OMC is enabled, an accumulator saturates to the most positive (0x7FFF_FFFF) or the most negative (0x8000_0000) number, depending on both the instruction and the value of the product that overflowed.</p> <p>1 Unsigned numbers. On overflow, if OMC is enabled, an accumulator saturates to the smallest value (0x0000_0000) or the largest value (0xFFFF_FFFF), depending on the instruction.</p> <p><b>In fractional mode:</b> S/U controls rounding while storing an accumulator to a general-purpose register.</p> <p>0 Move accumulator without rounding to a 16-bit value. Accumulator is moved to a general-purpose register as a 32-bit value.</p> <p>1 The accumulator is rounded to a 16-bit value using the round-to-nearest (even) method when it is moved to a general-purpose register. See <a href="#">Section 4.4.1.1.1, “Rounding.”</a> The resulting 16-bit value is stored in the lower word of the destination register. The upper word is zero-filled. The accumulator value is not affected by this rounding procedure.</p>
5	F/I	<p>Fractional/integer mode Determines whether input operands are treated as fractions or integers.</p> <p>0 Integers can be represented in either signed or unsigned notation, depending on the value of S/U.</p> <p>1 Fractions are represented in signed, fixed-point, two's complement notation. Values range from -1 to <math>1 - 2^{-15}</math> for 16-bit fractions and -1 to <math>1 - 2^{-31}</math> for 32-bit fractions. See <a href="#">Section 4.5.2, “Data Representation.”</a></p>

**Table 4-1. MACSR Field Descriptions (Continued)**

Bits	Name	Description
4	R/T	Round/truncate mode. Controls the rounding procedure for MOV.L ACCx,Rx, or MSAC.L instructions when operating in fractional mode. 0 Truncate. The product's lsbs are dropped before it is combined with the accumulator. Additionally, when a store accumulator instruction is executed (MOV.L ACCx,Rx), the 8 lsbs of the 48-bit accumulator logic are simply truncated. 1 Round-to-nearest (even). The 64-bit product of two 32-bit, fractional operands is rounded to the nearest 40-bit value. If the low-order 24 bits equal 0x80_0000, the upper 40 bits are rounded to the nearest even (lsb = 0) value. See <a href="#">Section 4.4.1.1.1, "Rounding."</a> Additionally, when a store accumulator instruction is executed (MOV.L ACCx,Rx), the lsbs of the 48-bit accumulator logic are used to round the resulting 16- or 32-bit value. If MACSR[S/U] = 0 and MACSR[R/T] = 1, the low-order 8 bits are used to round the resulting 32-bit fraction. If MACSR[S/U] = 1, the low-order 24 bits are used to round the resulting 16-bit fraction.
3–0	<b>Flags</b>	
3	N	Negative. Set if the msb of the result is set, otherwise cleared. N is affected only by MAC, MSAC, and load operations; it is not affected by MULS and MULU instructions.
2	Z	Zero. Set if the result equals zero, otherwise cleared. This bit is affected only by MAC, MSAC, and load operations; it is not affected by MULS and MULU instructions.
1	V	Overflow. Set if an arithmetic overflow occurs on a MAC or MSAC instruction indicating that the result cannot be represented in the limited width of the EMAC. V is set only if a product overflow occurs or the accumulation overflows the 48-bit structure. V is evaluated on each MAC or MSAC operation and uses the appropriate PAVx flag in the next-state V evaluation.
0	EV	Extension overflow. Signals that the last MAC or MSAC instruction overflowed the 32 lsbs in integer mode or the 40 lsbs in fractional mode of the destination accumulator. However, the result is still accurately represented in the combined 48-bit accumulator structure. Although an overflow has occurred, the correct result, sign, and magnitude are contained in the 48-bit accumulator. Subsequent MAC or MSAC operations may return the accumulator to a valid 32/40-bit result.

[Table 4-2](#) summarizes the interaction of the MACSR[S/U,F/I,R/T] control bits.

**Table 4-2. Summary of S/U, F/I, and R/T Control Bits**

S/U	F/I	R/T	Operational Modes
0	0	x	Signed, integer
0	1	0	Signed, fractional Truncate on MAC.L and MSAC.L No round on accumulator stores
0	1	1	Signed, fractional Round on MAC.L and MSAC.L Round-to-32-bits on accumulator stores
1	0	x	Unsigned, integer
1	1	0	Signed, fractional Truncate on MAC.L and MSAC.L Round-to-16-bits on accumulator stores
1	1	1	Signed, fractional Round on MAC.L and MSAC.L Round-to-16-bits on accumulator stores

#### 4.4.1.1 Fractional Operation Mode

This section describes behavior when the fractional mode is used (MACSR[F/I] is set).

##### 4.4.1.1.1 Rounding

When the processor is in fractional mode, there are two operations during which rounding can occur.

- Execution of a store accumulator instruction (MOV.L ACCx,Rx). The lsbs of the 48-bit accumulator logic are used to round the resulting 16- or 32-bit value. If MACSR[S/U] is cleared, the low-order 8 bits are used to round the resulting 32-bit fraction. If MACSR[S/U] is set, the low-order 24 bits are used to round the resulting 16-bit fraction.
- Execution of a MAC (or MSAC) instruction with 32-bit operands. If MACSR[R/T] is zero, multiplying two 32-bit numbers creates a 64-bit product that is truncated to the upper 40 bits; otherwise, it is rounded using round-to-nearest (even) method.

To understand the round-to-nearest-even method, consider the following example involving the rounding of a 32-bit number, R0, to a 16-bit number. Using this method, the 32-bit number is rounded to the closest 16-bit number possible. Let the high-order 16 bits of R0 be named R0.U and the low-order 16 bits be R0.L.

- If R0.L is less than 0x8000, the result is truncated to the value of R0.U.
- If R0.L is greater than 0x8000, the upper word is incremented (rounded up).
- If R0.L is 0x8000, R0 is half-way between two 16-bit numbers. In this case, rounding is based on the lsb of R0.U, so the result is always even (lsb = 0).
  - If the lsb of R0.U = 1 and R0.L = 0x8000, the number is rounded up.

— If the lsb of R0.U = 0 and R0.L = 0x8000, the number is rounded down.

This method minimizes rounding bias and creates as statistically correct an answer as possible.

The rounding algorithm is summarized in the following pseudocode:

```
if R0.L < 0x8000
    then Result = R0.U
    else if R0.L > 0x8000
        then Result = R0.U + 1
        else if lsb of R0.U = 0          /* R0.L = 0x8000 */
            then Result = R0.U
            else Result = R0.U + 1
```

The round-to-nearest-even technique is also known as convergent rounding.

#### 4.4.1.1.2 Saving and Restoring the EMAC Programming Model

The presence of rounding logic in the output datapath of the EMAC requires that special care be taken during the EMAC's save/restore process. In particular, any result rounding modes must be disabled during the save/restore process so the exact bit-wise contents of the EMAC registers are accessed. Consider the following memory structure containing the EMAC programming model:

```
struct macState {
    int acc0;
    int acc1;
    int acc2;
    int acc3;
    int accext01;
    int accext02;
    int mask;
    int macsr;
} macState;
```

The following assembly language routine shows the proper sequence for a correct EMAC state save. This code assumes all Dn and An registers are available for use and the memory location of the state save is defined by A7.

```
EMAC_state_save:
    move.l    macsr,d7          ; save the macsr
    clr.l     d0                ; zero the register to ...
    move.l    d0,macsr          ; disable rounding in the macsr
    move.l    acc0,d0           ; save the accumulators
    move.l    acc1,d1
    move.l    acc2,d2
    move.l    acc3,d3
    move.l    accext01,d4        ; save the accumulator extensions
    move.l    accext23,d5
    move.l    mask,d6           ; save the address mask
    movem.l   #0x00ff,(a7)      ; move the state to memory
```

The following code performs the EMAC state restore:

EMAC\_state\_restore:

```

    movem.l (a7),#0x00ff    ; restore the state from memory
    move.l  #0,macsr        ; disable rounding in the macsr
    move.l  d0,acc0         ; restore the accumulators
    move.l  d1,acc1
    move.l  d2,acc2
    move.l  d3,acc3
    move.l  d4,accest01     ; restore the accumulator extensions
    move.l  d5,accest23
    move.l  d6,mask         ; restore the address mask
    move.l  d7,macsr        ; restore the macsr

```

By executing this type of sequence, the exact state of the EMAC programming model can be correctly saved and restored.

#### 4.4.1.1.3 MULS/MULU

MULS and MULU are unaffected by fractional mode operation; operands are still assumed to be integers.

#### 4.4.1.1.4 Scale Factor in MAC or MSAC Instructions

The scale factor is ignored while the MAC is in fractional mode.

### 4.4.2 Mask Register (MASK)

The 32-bit MASK implements the low-order 16 bits to minimize the alignment complications involved with loading and storing only 16 bits. When the MASK is loaded, the low-order 16 bits of the source operand are actually loaded into the register. When it is stored, the upper 16 bits are all forced to ones.

This register performs a simple AND with the operand address for MAC instructions. That is, the processor calculates the normal operand address and, if enabled, that address is then ANDed with {0xFFFF, MASK[15:0]} to form the final address. Therefore, with certain MASK bits cleared, the operand address can be constrained to a certain memory region. This is used primarily to implement circular queues in conjunction with the (An)+ addressing mode.

This feature minimizes the addressing support required for filtering, convolution, or any routine that implements a data array as a circular queue. For MAC + MOVE operations, the MASK contents can optionally be included in all memory effective address calculations. The syntax is as follows:

MAC.sz Ry,RxSF,<ea>y&,Rw

The & operator enables the use of MASK and causes bit 5 of the extension word to be set. The exact algorithm for the use of MASK is as follows:

```

if extension word, bit [5] = 1, the MASK bit, then
    if <ea> = (An)
        oa = An & {0xFFFF, MASK}

    if <ea> = (An)+
        oa = An
        An = (An + 4) & {0xFFFF, MASK}

    if <ea> = -(An)
        oa = (An - 4) & {0xFFFF, MASK}
        An = (An - 4) & {0xFFFF, MASK}

    if <ea> = (d16,An)
        oa = (An + se_d16) & {0xFFFF0x, MASK}

```

Here, oa is the calculated operand address and se\_d16 is a sign-extended 16-bit displacement. For auto-addressing modes of post-increment and pre-decrement, the calculation of the updated An value is also shown.

Use of the post-increment addressing mode, {(An)+} with the MASK is suggested for circular queue implementations.

## 4.5 EMAC Instruction Set Summary

Table 4-3 summarizes EMAC unit instructions.

**Table 4-3. EMAC Instruction Summary**

Command	Mnemonic	Description
Multiply Signed	MULS <ea>y,Dx	Multiplies two signed operands yielding a signed result
Multiply Unsigned	MULU <ea>y,Dx	Multiplies two unsigned operands yielding an unsigned result
Multiply Accumulate	MAC Ry,RxSF,ACCx MSAC Ry,RxSF,ACCx	Multiplies two operands and adds/subtracts the product to/from an accumulator
Multiply Accumulate with Load	MAC Ry,Rx,<ea>y,Rw,ACCx MSAC Ry,Rx,<ea>y,Rw,ACCx	Multiplies two operands and combines the product to an accumulator while loading a register with the memory operand
Load Accumulator	MOV.L {Ry,#imm},ACCx	Loads an accumulator with a 32-bit operand
Store Accumulator	MOV.L ACCx,Rx	Writes the contents of an accumulator to a CPU register
Copy Accumulator	MOV.L ACCy,ACCx	Copies a 48-bit accumulator
Load MACSR	MOV.L {Ry,#imm},MACSR	Writes a value to MACSR
Store MACSR	MOV.L MACSR,Rx	Write the contents of MACSR to a CPU register
Store MACSR to CCR	MOV.L MACSR,CCR	Write the contents of MACSR to the CCR
Load MAC Mask Reg	MOV.L {Ry,#imm},MASK	Writes a value to the MASK register
Store MAC Mask Reg	MOV.L MASK,Rx	Writes the contents of the MASK to a CPU register
Load AccExtensions01	MOV.L {Ry,#imm},ACCext01	Loads the accumulator 0,1 extension bytes with a 32-bit operand



**Table 4-3. EMAC Instruction Summary (Continued)**

Command	Mnemonic	Description
Load AccExtensions23	MOV.L {Ry,#imm},ACCext23	Loads the accumulator 2,3 extension bytes with a 32-bit operand
Store AccExtensions01	MOV.L ACCext01,Rx	Writes the contents of accumulator 0,1 extension bytes into a CPU register
Store AccExtensions23	MOV.L ACCext23,Rx	Writes the contents of accumulator 2,3 extension bytes into a CPU register

### 4.5.1 EMAC Instruction Execution Times

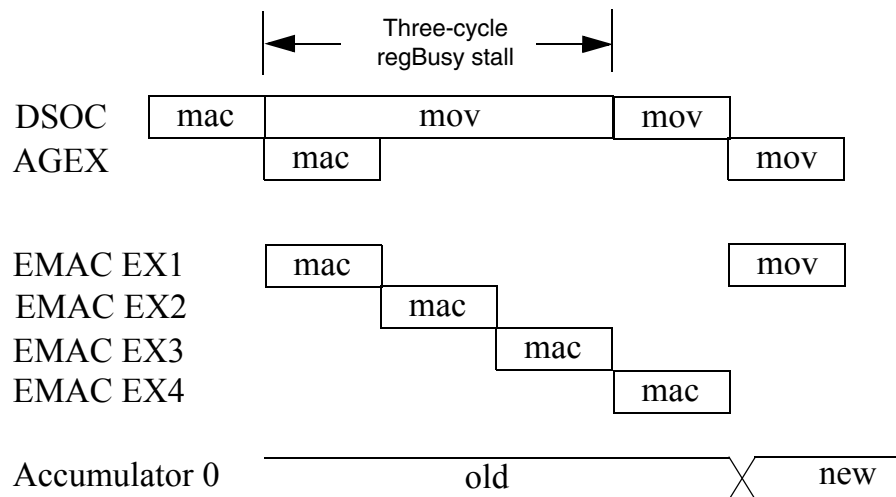
The instruction execution times for the EMAC can be found in [Section 3.12, “EMAC Instruction Execution Times.”](#)

The EMAC execution pipeline overlaps the AGEX stage of the OEP; that is, the first stage of the EMAC pipeline is the last stage of the basic OEP. EMAC units are designed for sustained, fully-pipelined operation on accumulator load, copy, and multiply-accumulate instructions. However, instructions that store contents of the multiply-accumulate programming model can generate OEP stalls that expose the EMAC execution pipeline depth, as in the following:

```
mac.w    Ry, Rx, Acc0
```

```
mov.l    Acc0, Rz
```

The mov.l instruction that stores the accumulator to an integer register (Rz) stalls until the program-visible copy of the accumulator is available. [Figure 4-8](#) shows EMAC timing.

**Figure 4-8. EMAC-Specific OEP Sequence Stall**

In [Figure 4-8](#), the OEP stalls the store-accumulator instruction for 3 cycles: the depth of the EMAC pipeline minus 1. The minus 1 factor is needed because the OEP and EMAC pipelines overlap by a cycle, the AGEX stage. As the store-accumulator instruction reaches the AGEX stage where the operation is performed, the just-updated accumulator 0 value is available.

As with change or use stalls between accumulators and general-purpose registers, introducing intervening instructions that do not reference the busy register can reduce or eliminate sequence-related store-MAC instruction stalls. In fact, a major benefit of the EMAC is the addition of three accumulators to minimize stalls caused by exchanges between the accumulator(s) and the general-purpose registers.

## 4.5.2 Data Representation

MACSR[S/U,F/I] selects one of the following three modes, where each mode defines a unique operand type.

- Two's complement signed integer: In this format, an N-bit operand value lies in the range  $-2^{(N-1)} \leq \text{operand} \leq 2^{(N-1)} - 1$ . The binary point is right of the lsb.
- Unsigned integer: In this format, an N-bit operand value lies in the range  $0 \leq \text{operand} \leq 2^N - 1$ . The binary point is right of the lsb.
- Two's complement, signed fractional: In an N-bit number, the first bit is the sign bit. The remaining bits signify the first N-1 bits after the binary point. Given an N-bit number,  $a_{N-1}a_{N-2}a_{N-3} \dots a_2a_1a_0$ , its value is given by the equation in [Figure 4-9](#).

$$\text{value} = -(1 \cdot a_{N-1}) + \sum_{i=0}^{N-2} 2^{(i+1-N)} \cdot a_i$$

**Figure 4-9. Two's Complement, Signed Fractional Equation**

This format can represent numbers in the range  $-1 \leq \text{operand} \leq 1 - 2^{(N-1)}$ .

For words and longwords, the largest negative number that can be represented is -1, whose internal representation is 0x8000 and 0x8000\_0000, respectively. The largest positive word is 0x7FFF or  $(1 - 2^{-15})$ ; the most positive longword is 0x7FFF\_FFFF or  $(1 - 2^{-31})$ .

## 4.5.3 MAC Opcodes

MAC opcodes are described in the *ColdFire Programmer's Reference Manual*.

Note the following:

- Unless otherwise noted, the value of MACSR[N,Z] is based on the result of the final operation that involves the product and the accumulator.

- The overflow (V) flag is handled differently. It is set if the complete product cannot be represented as a 40-bit value (this applies to  $32 \times 32$  integer operations only) or if the combination of the product with an accumulator cannot be represented in the given number of bits. The EMAC design includes an additional product/accumulation overflow bit for each accumulator that are treated as sticky indicators and are used to calculate the V bit on each MAC or MSAC instruction. See [Section 4.4.1, “MAC Status Register \(MACSR\).”](#)
- For the MAC design, the assembler syntax of the MAC (multiply and add to accumulator) and MSAC (multiply and subtract from accumulator) instructions does not include a reference to the single accumulator. For the EMAC, it is expected that assemblers support this syntax and that no explicit reference to an accumulator is interpreted as a reference to ACC0. These assemblers would also support syntaxes where the destination accumulator is explicitly defined.
- The optional 1-bit shift of the product is specified using the notation {<< | >>} SF, where <<1 indicates a left shift and >>1 indicates a right shift. The shift is performed before the product is added to or subtracted from the accumulator. Without this operator, the product is not shifted. If the EMAC is in fractional mode (MACSR[F/I] is set), SF is ignored and no shift is performed. Because a product can overflow, the following guidelines are implemented:
  - For unsigned word and longword operations, a zero is shifted into the product on right shifts.
  - For signed, word operations, the sign bit is shifted into the product on right shifts unless the product is zero. For signed, longword operations, the sign bit is shifted into the product unless an overflow occurs or the product is zero, in which case a zero is shifted in.
  - For all left shifts, a zero is inserted into the lsb position.

The following pseudocode explains basic MAC or MSAC instruction functionality. This example is presented as a case statement covering the three basic operating modes with signed integers, unsigned integers, and signed fractionals. Throughout this example, a comma-separated list in curly brackets, {}, indicates a concatenation operation.

```
switch (MACSR[6:5])      /* MACSR[S/U, F/I] */
{
  case 0:                /* signed integers */
    if (MACSR.OMC == 0 || MACSR.PAVx == 0)
      then {
        MACSR.PAVx = 0
        /* select the input operands */
        if (sz == word)
          then {if (U/Ly == 1)
            then operandY[31:0] = {sign-extended Ry[31], Ry[31:16]}
            else operandY[31:0] = {sign-extended Ry[15], Ry[15:0]}
            if (U/Lx == 1)
              then operandX[31:0] = {sign-extended Rx[31], Rx[31:16]}
```

```

        else operandX[31:0] = {sign-extended Rx[15], Rx[15:0]}
    }
    else {operandY[31:0] = Ry[31:0]
        operandX[31:0] = Rx[31:0]
    }

    /* perform the multiply */
    product[63:0] = operandY[31:0] * operandX[31:0]

    /* check for product overflow */
    if ((product[63:39] != 0x0000_00_0) && (product[63:39] != 0xffff_ff_1))
    then {
        /* product overflow */
        MACSR.PAVx = 1
        MACSR.V = 1
        if (inst == MSAC && MACSR.OMC == 1)
        then if (product[63] == 1)
            then result[47:0] = 0x0000_7fff_ffff
            else result[47:0] = 0xffff_8000_0000
        else if (MACSR.OMC == 1)
        then /* overflowed MAC,
            saturationMode enabled */
            if (product[63] == 1)
            then result[47:0] = 0xffff_8000_0000
            else result[47:0] = 0x0000_7fff_ffff
        }

    /* sign-extend to 48 bits before performing any scaling */
    product[47:40] = {8{product[39]}} /* sign-extend */

    /* scale product before combining with accumulator */
    switch (SF) /* 2-bit scale factor */
    {
        case 0: /* no scaling specified */
            break;
        case 1: /* SF = "<< 1" */
            product[40:0] = {product[39:0], 0}
            break;
        case 2: /* reserved encoding */
            break;
        case 3: /* SF = ">> 1" */
            product[39:0] = {product[39], product[39:1]}
            break;
    }

    if (MACSR.PAVx == 0)
    then {if (inst == MSAC)
        then result[47:0] = ACCx[47:0] - product[47:0]
        else result[47:0] = ACCx[47:0] + product[47:0]
    }

    /* check for accumulation overflow */
    if (accumulationOverflow == 1)
    then {MACSR.PAVx = 1

```

```

        MACSR.V = 1
        if (MACSR.OMC == 1)
            then /* accumulation overflow,
                  saturationMode enabled */
                if (result[47] == 1)
                    then result[47:0] = 0x0000_7fff_ffff
                    else result[47:0] = 0xffff_8000_0000
                }
        /* transfer the result to the accumulator */
        ACCx[47:0] = result[47:0]
    }
    MACSR.V = MACSR.PAVx
    MACSR.N = ACCx[47]
    if (ACCx[47:0] == 0x0000_0000_0000)
        then MACSR.Z = 1
        else MACSR.Z = 0
    if ((ACCx[47:31] == 0x0000_0) || (ACCx[47:31] == 0xffff_1))
        then MACSR.EV = 0
        else MACSR.EV = 1
break;
case 1,3:          /* signed fractionals */
if (MACSR.OMC == 0 || MACSR.PAVx == 0)
    then {
        MACSR.PAVx = 0
        if (sz == word)
            then {if (U/Ly == 1)
                    then operandY[31:0] = {Ry[31:16], 0x0000}
                    else operandY[31:0] = {Ry[15:0], 0x0000}
                if (U/Lx == 1)
                    then operandX[31:0] = {Rx[31:16], 0x0000}
                    else operandX[31:0] = {Rx[15:0], 0x0000}
                }
            else {operandY[31:0] = Ry[31:0]
                  operandX[31:0] = Rx[31:0]
                }
        }
        /* perform the multiply */
        product[63:0] = (operandY[31:0] * operandX[31:0]) << 1
        /* check for product rounding */
        if (MACSR.R/T == 1)
            then { /* perform convergent rounding */
                    if (product[23:0] > 0x80_0000)
                        then product[63:24] = product[63:24] + 1
                    else if ((product[23:0] == 0x80_0000) && (product[24] == 1))
                        then product[63:24] = product[63:24] + 1
                }
        /* sign-extend to 48 bits and combine with accumulator */
        /* check for the -1 * -1 overflow case */
        if ((operandY[31:0] == 0x8000_0000) && (operandX[31:0] == 0x8000_0000))
            then product[71:64] = 0x00 /* zero-fill */
            else product[71:64] = {8{product[63]}} /* sign-extend */
        if (inst == MSAC)
            then result[47:0] = ACCx[47:0] - product[71:24]
            else result[47:0] = ACCx[47:0] + product[71:24]
    }

```

```

        /* check for accumulation overflow */
        if (accumulationOverflow == 1)
            then {MACSR.PAVx = 1
                MACSR.V = 1
                if (MACSR.OMC == 1)
                    then /* accumulation overflow,
                        saturationMode enabled */
                        if (result[47] == 1)
                            then result[47:0] = 0x007f_ffff_ff00
                            else result[47:0] = 0xff80_0000_0000
                        }
            }
        /* transfer the result to the accumulator */
        ACCx[47:0] = result[47:0]
    }
    MACSR.V = MACSR.PAVx
    MACSR.N = ACCx[47]
    if (ACCx[47:0] == 0x0000_0000_0000)
        then MACSR.Z = 1
        else MACSR.Z = 0
    if ((ACCx[47:39] == 0x00_0) || (ACCx[47:39] == 0xff_1))
        then MACSR.EV = 0
        else MACSR.EV = 1
break;
case 2:                /* unsigned integers */
    if (MACSR.OMC == 0 || MACSR.PAVx == 0)
        then {
            MACSR.PAVx = 0
            /* select the input operands */
            if (sz == word)
                then {if (U/Ly == 1)
                    then operandY[31:0] = {0x0000, Ry[31:16]}
                    else operandY[31:0] = {0x0000, Ry[15:0]}
                    if (U/Lx == 1)
                        then operandX[31:0] = {0x0000, Rx[31:16]}
                        else operandX[31:0] = {0x0000, Rx[15:0]}
                    }
                else {operandY[31:0] = Ry[31:0]
                    operandX[31:0] = Rx[31:0]
                }
            }

            /* perform the multiply */
            product[63:0] = operandY[31:0] * operandX[31:0]

            /* check for product overflow */
            if (product[63:40] != 0x0000_00)
                then {
                    /* product overflow */
                    MACSR.PAVx = 1
                    MACSR.V = 1
                    if (inst == MSAC && MACSR.OMC == 1)
                        then result[47:0] = 0x0000_0000_0000
                    else if (MACSR.OMC == 1)
                        then /* overflowed MAC,
                            saturationMode enabled */

```

```

        result[47:0] = 0xffff_ffff_ffff

    }

    /* zero-fill to 48 bits before performing any scaling */
    product[47:40] = 0    /* zero-fill upper byte */

    /* scale product before combining with accumulator */
    switch (SF)    /* 2-bit scale factor */
    {
        case 0:    /* no scaling specified */
            break;
        case 1:    /* SF = "<< 1" */
            product[40:0] = {product[39:0], 0}
            break;
        case 2:    /* reserved encoding */
            break;
        case 3:    /* SF = ">> 1" */
            product[39:0] = {0, product[39:1]}
            break;
    }

    /* combine with accumulator */
    if (MACSR.PAVx == 0)
    then {if (inst == MSAC)
        then result[47:0] = ACCx[47:0] - product[47:0]
        else result[47:0] = ACCx[47:0] + product[47:0]
    }

    /* check for accumulation overflow */
    if (accumulationOverflow == 1)
    then {MACSR.PAVx = 1
        MACSR.V = 1
        if (inst == MSAC && MACSR.OMC == 1)
        then result[47:0] = 0x0000_0000_0000
        else if (MACSR.OMC == 1)
        then /* overflowed MAC,
            saturationMode enabled */
            result[47:0] = 0xffff_ffff_ffff
    }

    /* transfer the result to the accumulator */
    ACCx[47:0] = result[47:0]
}
MACSR.V = MACSR.PAVx
MACSR.N = ACCx[47]
if (ACCx[47:0] == 0x0000_0000_0000)
then MACSR.Z = 1
else MACSR.Z = 0
if (ACCx[47:32] == 0x0000)
then MACSR.EV = 0
else MACSR.EV = 1
break;
}

```





# Chapter 5

## Cache

### 5.1 Introduction

This chapter describes the MCF5275 cache operation.

#### 5.1.1 Features

- Configurable as instruction, data, or split instruction/data cache
- 16-Kbyte direct-mapped cache
- Single-cycle access on cache hits
- Physically located on the Coldfire core's high-speed local bus
- Nonblocking design to maximize performance
- Separate instruction and data 16-Byte line-fill buffers
- Configurable instruction cache miss-fetch algorithm

#### 5.1.2 Physical Organization

The cache is a direct-mapped single-cycle memory. It may be configured as an instruction cache, a write-through data cache, or a split instruction/data cache. The cache storage is organized as 1024 lines, each containing 16 bytes. The memory storage consists of a 1024-entry tag array (containing addresses and a valid bit), and a data array containing 16 Kbytes, organized as  $4096 \times 32$  bits.

Cache configuration is controlled by bits in the cache control register (CACR) that is detailed later in this chapter. For the instruction or data-only configurations, only the associated instruction or data line-fill buffer is used. For the split cache configuration, one-half of the tag and storage arrays is used for an instruction cache and one-half is used for a data cache. The split cache configuration uses both the instruction and the data line-fill buffers. The core's local bus is a unified bus used for both instruction and data fetches. Therefore, the cache can have only one fetch, either instruction or data, active at one time.

For the instruction- or data-only configurations, the cache tag and storage arrays are accessed in parallel: fetch address bits [13:4] addressing the tag array and fetch address bits [13:2] addressing the storage array. For the split cache configuration, the cache tag and storage arrays are accessed in parallel. The msb of the tag array address is set for instruction fetches and cleared for operand fetches; fetch address bits [12:4] provide the rest of the tag array address. The tag array outputs the address mapped to the given cache location along with the valid bit for the line. This address field is compared to bits [31:14] for instruction- or data-only configurations and to bits [31:13] for a

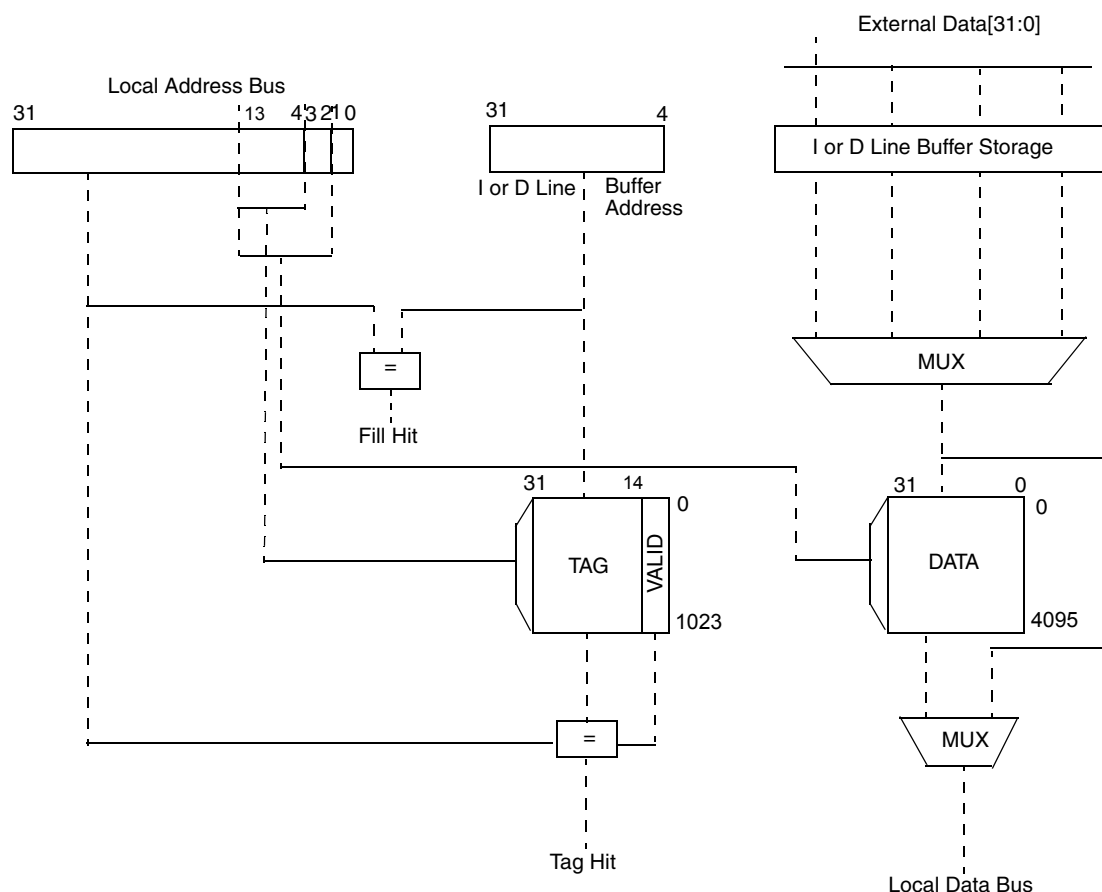
split configuration of the fetch address from the local bus to determine if a cache hit has occurred. If the desired address is mapped into the cache memory, the output of the storage array is driven onto the ColdFire core's local data bus, thereby completing the access in a single cycle.

The tag array maintains a single valid bit per line entry. Accordingly, only entire 16-byte lines are loaded into the cache.

The cache also contains separate 16-byte instruction and data line-fill buffers that provide temporary storage for the last line fetched in response to a cache miss. With each fetch, the contents of the associated line fill buffer are examined. Thus, each fetch address examines both the tag memory array and the associated line fill buffer to see if the desired address is mapped into either hardware resource. A cache hit in either the memory array or the associated line-fill buffer is serviced in a single cycle. Because the line fill buffer maintains valid bits on a longword basis, hits in the buffer can be serviced immediately without waiting for the entire line to be fetched.

If the referenced address is not contained in the memory array or the associated line-fill buffer, the cache initiates the required external fetch operation. In most situations, this is a 16-byte line-sized burst reference.

The hardware implementation is a nonblocking design, meaning the ColdFire core's local bus is released after the initial access of a miss. Thus, the cache or the SRAM module can service subsequent requests while the remainder of the line is being fetched and loaded into the fill buffer.



**Figure 5-1. Cache Block Diagram**

### 5.1.3 Operation

The cache is physically connected to the ColdFire core's local bus, allowing it to service all fetches from the ColdFire core and certain memory fetches initiated by the debug module. Typically, the debug module's memory references appear as supervisor data accesses but the unit can be programmed to generate user-mode accesses and/or instruction fetches. The cache processes any fetch access in the normal manner.

#### 5.1.3.1 Interaction with Other Modules

Because both the cache and high-speed SRAM module are connected to the ColdFire core's local data bus, certain user-defined configurations can result in simultaneous fetch processing.

If the referenced address is mapped into the SRAM module, that module will service the request in a single cycle. In this case, data accessed from the cache is simply discarded and no external memory references are generated. If the address is not mapped into the SRAM space, the cache handles the request in the normal fashion.

### 5.1.3.2 Memory Reference Attributes

For every memory reference the ColdFire core or the debug module generates, a set of “effective attributes” is determined based on the address and the access control registers (ACRs). This set of attributes includes the cacheable/noncacheable definition, the precise/imprecise handling of operand write, and the write-protect capability.

In particular, each address is compared to the values programmed in the ACRs. If the address matches one of the ACR values, the access attributes from that ACR are applied to the reference. If the address does not match either ACR, then the default value defined in the cache control register (CACR) is used. The specific algorithm is as follows:

```
if (address == ACR0_address including mask)
    Effective Attributes = ACR0 attributes
else if (address == ACR1_address including mask)
    Effective Attributes = ACR1 attributes
else Effective Attributes = CACR default attributes
```

### 5.1.3.3 Cache Coherency and Invalidation

The cache does not monitor ColdFire core data references for accesses to cached instructions. Therefore, software must maintain instruction cache coherency by invalidating the appropriate cache entries after modifying code segments if instructions are cached.

The cache invalidation can be performed in several ways. For the instruction- or data-only configurations, setting CACR[CINV] forces the entire cache to be marked as invalid. The invalidation operation requires 1024 cycles because the cache sequences through the entire tag array, clearing a single location each cycle. For the split configuration, CACR[INVI] and CACR[INVD] can be used in addition to CACR[CINV] to clear the entire cache, only the instruction half, or only the data half. Any subsequent fetch accesses are postponed until the invalidation sequence is complete.

The privileged CPUSHL instruction can invalidate a single cache line. When this instruction is executed, the cache entry defined by bits [13:4] of the source address register is invalidated, provided CACR[CPDI] is cleared. For the split data/instruction cache configuration, software directly controls bit 13 which selects whether an instruction cache or data cache line is being accessed.

These invalidation operations can be initiated from the ColdFire core or the debug module.

### 5.1.3.4 Reset

A hardware reset clears the CACR and disables the cache. The contents of the tag array are not affected by the reset. Accordingly, the system startup code must explicitly perform a cache invalidation by setting CACR[CINV] before the cache can be enabled.

### 5.1.3.5 Cache Miss Fetch Algorithm/Line Fills

As discussed in [Section 5.1.2, “Physical Organization,”](#) the cache hardware includes a 16-byte line-fill buffer for providing temporary storage for the last fetched line.

With the cache enabled as defined by CACR[CENB], a cacheable fetch that misses in both the tag memory and the line-fill buffer generates an external fetch. For data misses, the size of the external fetch is always 16 bytes. For instruction misses, the size of the external fetch is determined by the value contained in the 2-bit CLNF field of the CACR and the miss address. [Table 5-1](#) shows the relationship between the CLNF bits, the miss address, and the size of the external fetch.

**Table 5-1. Initial Fetch Offset vs. CLNF Bits**

CLNF[1:0] ]	Longword Address Bits			
	00	01	10	11
00	Line	Line	Line	Longword
01	Line	Line	Longword	Longword
1X	Line	Line	Line	Line

Depending on the runtime characteristics of the application and the memory response speed, overall performance may be increased by programming the CLNF bits to values {00, 01}.

For all cases of a line-sized fetch, the critical longword defined by bits [3:2] of the miss address is accessed first followed by the remaining three longwords that are accessed by incrementing the longword address in a modulo-16 fashion as shown below:

```

if miss address[3:2] = 00
    fetch sequence = {0x0, 0x4, 0x8, 0xC}

if miss address[3:2] = 01
    fetch sequence = {0x4, 0x8, 0xC, 0x0}

if miss address[3:2] = 10
    fetch sequence = {0x8, 0xC, 0x0, 0x4}

if miss address[3:2] = 11
    fetch sequence = {0xC, 0x0, 0x4, 0x8}

```

Once an external fetch has been initiated and the data is loaded into the line-fill buffer, the cache maintains a special “most-recently-used” indicator that tracks the contents of the associated

line-fill buffer versus its corresponding cache location. At the time of the miss, the hardware indicator is set, marking the line-fill buffer as “most recently used.” If a subsequent access occurs to the cache location defined by bits [13:4] (or bits [12:4] for split configurations of the fill buffer address), the data in the cache memory array is now most recently used, so the hardware indicator is cleared. In all cases, the indicator defines whether the contents of the line-fill buffer or the memory data array are most recently used. At the time of the next cache miss, the contents of the line-fill buffer are written into the memory array if the entire line is present, and the line-fill buffer data is still most recently used compared to the memory array.

Generally, longword references are used for sequential instruction fetches. If the processor branches to an odd word address, a word-sized instruction fetch is generated.

For instruction fetches, the fill buffer can also be used as temporary storage for line-sized bursts of non-cacheable references under control of CACR[CEIB]. With this bit set, a noncacheable instruction fetch is processed as defined by [Table 5-2](#). For this condition, the line-fill buffer is loaded and subsequent references can hit in the buffer, but the data is never loaded into the memory array.

[Table 5-2](#) shows the relationship between CACR bits CENB and CEIB and the type of instruction fetch.

**Table 5-2. Instruction Cache Operation as Defined by CACR**

CACR [CENB]	CACR [CEIB]	Type of Instruction Fetch	Description
0	0	N/A	Cache is completely disabled; all instruction fetches are word or longword in size.
0	1	N/A	All instruction fetches are word or longword in size
1	X	Cacheable	Fetch size is defined by <a href="#">Table 5-1</a> and contents of the line-fill buffer can be written into the memory array
1	0	Noncacheable	All instruction fetches are word or longword in size, and not loaded into the line-fill buffer
1	1	Noncacheable	Instruction fetch size is defined by <a href="#">Table 5-1</a> and loaded into the line-fill buffer, but are never written into the memory array.

## 5.2 Memory Map/Register Definition

Three supervisor registers define the operation of the cache and local bus controller: the cache control register (CACR) and two access control registers (ACR0, ACR1). [Table 5-3](#) below shows the memory map of the cache and access control registers.

The following lists several keynotes regarding the programming model table:

- The CACR and ACRs can only be accessed in supervisor mode using the MOVEC instruction with an Rc value of 0x002, 0x004 and 0x005, respectively.

- Addresses not assigned to the registers and undefined register bits are reserved for future expansion. The user should write zeros to these reserved address spaces and read accesses will return zeros.
- The reset value column indicates the register initial value at reset. Certain registers may be uninitialized upon reset; that is, they may contain random values after reset.
- The access column indicates if the corresponding register allows both read/write functionality (R/W), read-only functionality (R), or write-only functionality (W). If a read access to a write-only register is attempted, zeros will be returned. If a write access to a read-only register is attempted, the access will be ignored and no write will occur.

**Table 5-3. Memory Map of Cache Registers**

Address	Name	Width	Description	Reset Value	Access <sup>1</sup>
MOVEC with 0x002	CACR	32	Cache Control Register	0x0000_0000	W
MOVEC with 0x004	ACR0	32	Access Control Register 0	0x0000_0000	W
MOVEC with 0x005	ACR1	32	Access Control Register 1	0x0000_0000	W
<sup>1</sup> Readable through debug					

## 5.2.1 Registers Description

### 5.2.1.1 Cache Control Register (CACR)

The CACR controls the operation of the cache. The CACR provides a set of default memory access attributes used when a reference address does not map into the spaces defined by the ACRs.

The CACR is a 32-bit write-only supervisor control register. It is accessed in the CPU address space via the MOVEC instruction with an Rc encoding of 0x002. The CACR can be read when in background debug mode (BDM). At system reset, the entire register is cleared.

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R																
W	CENB	—		CPD	CFRZ	—		CINV	DISI	DISD	INVI	INVD	—			
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																
W	—					CEIB	DCM	DBWE	—		DWP	EUSP	—		CLNF	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Address	MOVEC with 0x002															

**Figure 5-2. Cache Control Register (CACR)**

**Table 5-4. CACR Field Descriptions**

Bits	Name	Description
31	CENB	Cache enable. The memory array of the cache is enabled only if CENB is asserted. This bit, along with the DISI (disable instruction caching) and DISD (disable data caching) bits, control the cache configuration. 0 Cache disabled 1 Cache enabled <a href="#">Table 5-5</a> describes cache configuration.
30–29	—	Reserved, should be cleared.
28	CPDI	Disable CPUSHL invalidation. When the privileged CPUSHL instruction is executed, the cache entry defined by bits [13:4] of the address is invalidated if CPDI = 0. If CPDI = 1, no operation is performed. 0 Enable invalidation 1 Disable invalidation
27	CFRZ	Cache freeze. This field allows the user to freeze the contents of the cache. When CFRZ is asserted line fetches can be initiated and loaded into the line-fill buffer, but a valid cache entry can not be overwritten. If a given cache location is invalid, the contents of the line-fill buffer can be written into the memory array while CFRZ is asserted. 0 Normal Operation 1 Freeze valid cache lines
26–25	—	Reserved, should be cleared.
24	CINV	Cache invalidate. The cache invalidate operation is not a function of the CENB state (that is, this operation is independent of the cache being enabled or disabled). Setting this bit forces the cache to invalidate all, half, or none of the tag array entries depending on the state of the DISI, DISD, INVI, and INVD bits. The invalidation process requires several cycles of overhead plus 1024 machine cycles to clear all tag array entries and 64512 cycles to clear half of the tag array entries, with a single cache entry cleared per machine cycle. The state of this bit is always read as a zero. After a hardware reset, the cache must be invalidated before it is enabled. 0 No operation 1 Invalidate all cache locations <a href="#">Table 5-6</a> describes how to set the cache invalidate all bit.
23	DISI	Disable instruction caching. When set, this bit disables instruction caching. This bit, along with the CENB (cache enable) and DISD (disable data caching) bits, control the cache configuration. See the CENB definition for a detailed description. 0 Do not disable instruction caching 1 Disable instruction caching <a href="#">Table 5-5</a> describes cache configuration and <a href="#">Table 5-6</a> describes how to set the cache invalidate all bit.
22	DISD	Disable data caching. When set, this bit disables data caching. This bit, along with the CENB (cache enable) and DISI (disable instruction caching) bits, control the cache configuration. See the CENB definition for a detailed description. 0 Do not disable data caching 1 Disable data caching <a href="#">Table 5-5</a> describes cache configuration and <a href="#">Table 5-6</a> describes how to set the cache invalidate all bit.



**Table 5-4. CACR Field Descriptions (Continued)**

Bits	Name	Description
21	INVI	CINV instruction cache only. This bit can not be set unless the cache configuration is split (both DISI and DISD cleared). For instruction or data cache configurations this bit is a don't-care. For the split cache configuration, this bit is part of the control for the invalidate all operation. See the CINV definition for a detailed description <a href="#">Table 5-6</a> describes how to set the cache invalidate all bit.
20	INVD	CINV data cache only. This bit can not be set unless the cache configuration is split (both DISI and DISD cleared). For instruction or data cache configurations this bit is a don't-care. For the split cache configuration, this bit is part of the control for the invalidate all operation. See the CINV definition for a detailed description <a href="#">Table 5-6</a> describes how to set the cache invalidate all bit.
19–11	—	Reserved, should be cleared.
10	CEIB	Cache enable noncacheable instruction bursting. Setting this bit enables the line-fill buffer to be loaded with burst transfers under control of CLNF[1:0] for noncacheable accesses. Noncacheable accesses are never written into the memory array. See <a href="#">Table 5-2</a> . 0 Disable burst fetches on noncacheable accesses 1 Enable burst fetches on noncacheable accesses
9	DCM	Default cache mode. This bit defines the default cache mode: 0 is cacheable, 1 is noncacheable. For more information on the selection of the effective memory attributes, see <a href="#">Section 5.1.3.2, "Memory Reference Attributes</a> . 0 Caching enabled 1 Caching disabled
8	DBWE	Default buffered write enable. This bit defines the default value for enabling buffered writes. If DBWE = 0, the termination of an operand write cycle on the processor's local bus is delayed until the external bus cycle is completed. If DBWE = 1, the write cycle on the local bus is terminated immediately and the operation buffered in the bus controller. In this mode, operand write cycles are effectively decoupled between the processor's local bus and the external bus. Generally, enabled buffered writes provide higher system performance but recovery from access errors can be more difficult. For the ColdFire core, reporting access errors on operand writes is always imprecise and enabling buffered writes further decouples the write instruction and the signaling of the fault 0 Disable buffered writes 1 Enable buffered writes
7–6	—	Reserved, should be cleared.
5	DWP	Default write protection 0 Read and write accesses permitted 1 Only read accesses permitted
4	EUSP	Enable user stack pointer. See <a href="#">Section 3.2.3.2, "Supervisor/User Stack Pointers (A7 and OTHER_A7),"</a> for more information on the dual stack pointer implementation. 0 Disable the processor's use of the User Stack Pointer 1 Enable the processor's use of the User Stack Pointer
3–2	—	Reserved, should be cleared.
1–0	CLNF	Cache line fill. These bits control the size of the memory request the cache issues to the bus controller for different initial instruction line access offsets. See <a href="#">Table 5-1</a> for external fetch size based on miss address and CLNF.

Table 5-5 shows the relationship between CACR bits CENB, DISI, & DISD and the cache configuration.

**Table 5-5. Cache Configuration as Defined by CACR**

CACR [CENB]	CACR [DISI]	CACR [DISD]	Configuration	Description
0	x	x	N/A	Cache is completely disabled
1	0	0	Split Instruction/ Data Cache	8 KByte direct-mapped instruction cache (uses lower half of tag and storage arrays) and 8 KByte direct-mapped write-through data cache (uses upper half of tag and storage arrays)
1	0	1	Instruction Cache	16 KByte direct-mapped instruction cache (uses all of tag and storage arrays)
1	1	0	Data Cache	16 KByte direct-mapped write-through data cache (uses all of tag and storage arrays)

Table 5-6 shows the relationship between CACR bits DISI, DISD, INVI, & INVD and setting the cache invalidate all bit.

**Table 5-6. Cache Invalidate All as Defined by CACR**

CACR [DISI]	CACR [DISD]	CACR [INVI]	CACR [INVD]	Configuration	Operation
0	0	0	0	Split Instruction/ Data Cache	Invalidate all entries in both 8 KByte instruction cache and 8 KByte data cache
0	0	0	1	Split Instruction/ Data Cache	Invalidate only 8 KByte data cache
0	0	1	0	Split Instruction/ Data Cache	Invalidate only 8 KByte instruction cache
0	0	1	1	Split Instruction/ Data Cache	No invalidate
1	0	x	x	Instruction Cache	Invalidate 16 KByte instruction cache
0	1	x	x	Data Cache	Invalidate 16 KByte data cache

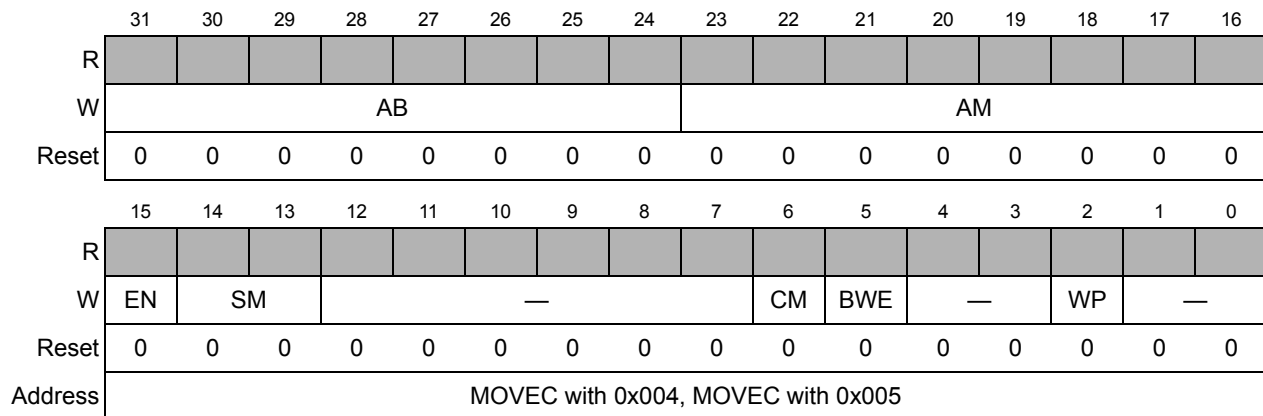
### 5.2.1.2 Access Control Registers (ACR0, ACR1)

The ACRs provide a definition of memory reference attributes for two memory regions (one per ACR). This set of effective attributes is defined for every memory reference using the ACRs or the set of default attributes contained in the CACR. The ACRs are examined for every processor memory reference that is not mapped to the SRAM memories.

The ACRs are 32-bit write-only supervisor control register. They are accessed in the CPU address space via the MOVEC instruction with an Rc encoding of 0x004 and 0x005. The ACRs can be read when in background debug mode (BDM). At system reset, both registers are cleared.

**NOTE**

IPSBAR space cannot be cached. Ensure that ACR[AB] does not fall within this space.

**Figure 5-3. Access Control Registers (ACR0, ACR1)****Table 5-7. ACR Field Descriptions**

Bits	Name	Description
31–24	AB	Address base. This 8-bit field is compared to address bits [31:24] from the processor's local bus under control of the ACR address mask. If the address matches, the attributes for the memory reference are sourced from the given ACR.
23–16	AM	Address mask. This 8-bit field can mask any bit of the AB field comparison. If a bit in the AM field is set, then the corresponding bit of the address field comparison is ignored.
15	EN	ACR Enable. The EN bit defines the ACR enable. Hardware reset clears this bit, disabling the ACR. 0 ACR disabled 1 ACR enabled
14–13	SM	Supervisor mode. This two-bit field allows the given ACR to be applied to references based on operating privilege mode of the ColdFire processor. The field uses the ACR for user references only, supervisor references only, or all accesses. 00 Match if user mode 01 Match if supervisor mode 1x Match always—ignore user/supervisor mode
12–7	—	Reserved, should be cleared.
6	CM	Cache mode. This bit defines the cache mode: 0 is cacheable, 1 is noncacheable. 0 Caching enabled 1 Caching disabled

**Table 5-7. ACR Field Descriptions (Continued)**

Bits	Name	Description
5	BWE	<p>Buffered write enable. This bit defines the value for enabling buffered writes. If BWE = 0, the termination of an operand write cycle on the processor's local bus is delayed until the external bus cycle is completed. If BWE = 1, the write cycle on the local bus is terminated immediately and the operation is then buffered in the bus controller. In this mode, operand write cycles are effectively decoupled between the processor's local bus and the external bus.</p> <p>Generally, the enabling of buffered writes provides higher system performance but recovery from access errors may be more difficult. For the V2 ColdFire core, the reporting of access errors on operand writes is always imprecise, and enabling buffered writes simply decouples the write instruction from the signaling of the fault even more.</p> <p>0 Writes are not buffered. 1 Writes are buffered.</p>
4–3	—	Reserved, should be cleared.
2	WP	<p>Write protect. The WP bit defines the write-protection attribute. If the effective memory attributes for a given access select the WP bit, an access error terminates any attempted write with this bit set.</p> <p>0 Read and write accesses permitted 1 Only read accesses permitted</p>
1–0	—	Reserved, should be cleared.

# Chapter 6

## Static RAM (SRAM)

### 6.1 Introduction

This chapter is a description of the on-chip static RAM (SRAM) implementation that covers general operations, configuration, and initialization. It also provides information and examples showing how to minimize power consumption when using the SRAM.

#### 6.1.1 Features

- One 64-Kbyte SRAM
- Single-cycle access
- Physically located on processor's high-speed local bus
- Memory location programmable on any 0-modulo-64 Kbyte address
- Byte, word, longword address capabilities

#### 6.1.2 Operation

The SRAM module provides a general-purpose memory block that the ColdFire processor can access in a single cycle. The location of the memory block can be specified to any 0-modulo-64K address within the 4-Gbyte address space. The memory is ideal for storing critical code or data structures or for use as the system stack. Because the SRAM module is physically connected to the processor's high-speed local bus, it can service processor-initiated access or memory-referencing commands from the debug module.

Depending on configuration information, instruction fetches may be sent to both the cache and the SRAM block simultaneously. If the reference is mapped into the region defined by the SRAM, the SRAM provides the data back to the processor, and the cache data is discarded. Accesses from the SRAM module are not cached.

The SRAM is dual-ported to provide DMA or FEC access. The SRAM is partitioned into two physical memory arrays to allow simultaneous access to both arrays by the processor core and another bus master. See [Section 11.3, “Internal Bus Arbitration,”](#) for more information.

### 6.2 Register Description

The SRAM programming model includes a description of the SRAM base address register (RAMBAR), SRAM initialization, and power management.

## 6.2.1 SRAM Base Address Register (RAMBAR)

The configuration information in the SRAM base address register (RAMBAR) controls the operation of the SRAM module.

- The RAMBAR holds the base address of the SRAM. The MOVEC instruction provides write-only access to this register.
- The RAMBAR can be read or written from the debug module in a similar manner.
- All undefined bits in the register are reserved. These bits are ignored during writes to the RAMBAR, and return zeroes when read from the debug module.
- The RAMBAR valid bit is cleared by reset, disabling the SRAM module. All other bits are unaffected.

### NOTE

Do not confuse this RAMBAR with the SCM RAMBAR in [Section 11.2.1.2, “Memory Base Address Register \(RAMBAR\).”](#) Although similar, this core RAMBAR enables core access to the SRAM memory, while the SCM RAMBAR enables peripheral (e.g. DMA and FEC) access to the SRAM.

The RAMBAR contains several control fields. These fields are shown in [Figure 6-1](#).

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	See Note															
W	BA															
Reset	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	See Note															
W	0	0	0	0	PRI1	PRI0	SPV	WP	0	0	C/I	SC	SD	UC	UD	V
Reset	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	0
Address	CPU + 0x0C05															

Note: W for Core; R/W for Debug

**Figure 6-1. SRAM Base Address Register (RAMBAR)**

**Table 6-1. RAMBAR Field Descriptions**

Bits	Name	Description
31–16	BA	Base address. Defines the 0-modulo-64K base address of the SRAM module. By programming this field, the SRAM may be located on any 64-Kbyte boundary within the processor's 4-Gbyte address space.
15–12	—	Reserved, should be cleared.

Table 6-1. RAMBAR Field Descriptions (Continued)

Bits	Name	Description															
11–10	PRI1 PRI0	<p>Priority bit. PRI1 determines if DMA/FEC or CPU has priority in upper 32k bank of memory. PRI0 determines if DMA/FEC or CPU has priority in lower 32k bank of memory. If bit is set, DMA/FEC has priority. If bit is cleared, CPU has priority. Priority is determined according to the following table.</p> <table border="1"> <thead> <tr> <th>PRI[1:0]</th><th>Upper Bank Priority</th><th>Lower Bank Priority</th></tr> </thead> <tbody> <tr> <td>00</td><td>CPU Accesses</td><td>CPU Accesses</td></tr> <tr> <td>01</td><td>CPU Accesses</td><td>DMA/FEC Accesses</td></tr> <tr> <td>10</td><td>DMA/FEC Accesses</td><td>CPU Accesses</td></tr> <tr> <td>11</td><td>DMA/FEC Accesses</td><td>DMA/FEC Accesses</td></tr> </tbody> </table> <p><b>Note:</b> The recommended setting for the priority bits is 00.</p>	PRI[1:0]	Upper Bank Priority	Lower Bank Priority	00	CPU Accesses	CPU Accesses	01	CPU Accesses	DMA/FEC Accesses	10	DMA/FEC Accesses	CPU Accesses	11	DMA/FEC Accesses	DMA/FEC Accesses
PRI[1:0]	Upper Bank Priority	Lower Bank Priority															
00	CPU Accesses	CPU Accesses															
01	CPU Accesses	DMA/FEC Accesses															
10	DMA/FEC Accesses	CPU Accesses															
11	DMA/FEC Accesses	DMA/FEC Accesses															
9	SPV	<p>Secondary port valid. Allows access by DMA and FEC</p> <p>0 DMA and FEC access to memory is disabled.</p> <p>1 DMA and FEC access to memory is enabled.</p> <p><b>Note:</b> The BDE bit in the second RAMBAR register must also be set to allow dual port access to the SRAM. For more information, see <a href="#">Section 11.2.1.2, “Memory Base Address Register (RAMBAR).”</a></p>															
8	WP	<p>Write protect. Allows only read accesses to the SRAM. When this bit is set, any attempted write access will generate an access error exception to the ColdFire processor core.</p> <p>0 Allows read and write accesses to the SRAM module</p> <p>1 Allows only read accesses to the SRAM module</p>															
7–6	—	Reserved, should be cleared.															
5–1	C/I, SC, SD, UC, UD	<p>Address space masks (ASn)</p> <p>These five bit fields allow certain types of accesses to be “masked,” or inhibited from accessing the SRAM module. The address space mask bits are:</p> <p>C/I = CPU space/interrupt acknowledge cycle mask</p> <p>SC = Supervisor code address space mask</p> <p>SD = Supervisor data address space mask</p> <p>UC = User code address space mask</p> <p>UD = User data address space mask</p> <p>For each address space bit:</p> <p>0 An access to the SRAM module can occur for this address space</p> <p>1 Disable this address space from the SRAM module. If a reference using this address space is made, it is inhibited from accessing the SRAM module, and is processed like any other non-SRAM reference.</p> <p>These bits are useful for power management as detailed in <a href="#">Section 6.2.4, “Power Management.”</a></p>															
0	V	<p>Valid. A hardware reset clears this bit. When set, this bit enables the SRAM module; otherwise, the module is disabled.</p> <p>0 Contents of RAMBAR are not valid</p> <p>1 Contents of RAMBAR are valid</p>															

## 6.2.2 SRAM Initialization

After a hardware reset, the contents of the SRAM module are undefined. The valid bit of the RAMBAR is cleared, disabling the module. If the SRAM requires initialization with instructions or data, the following steps should be performed:

1. Load the RAMBAR mapping the SRAM module to the desired location within the address space.
2. Read the source data and write it to the SRAM. There are various instructions to support this function, including memory-to-memory move instructions, or the MOVEM opcode. The MOVEM instruction is optimized to generate line-sized burst fetches on 0-modulo-16 addresses, so this opcode generally provides maximum performance.
3. After the data has been loaded into the SRAM, it may be appropriate to load a revised value into the RAMBAR with a new set of attributes. These attributes consist of the write-protect and address space mask fields.

The ColdFire processor or an external emulator using the debug module can perform these initialization functions.

## 6.2.3 SRAM Initialization Code

The following code segment describes how to initialize the SRAM. The code sets the base address of the SRAM at 0x2000\_0000 and then initializes the SRAM to zeros.

```
RAMBASE      EQU $20000000      ;set this variable to $20000000
RAMVALID     EQU $00000001
move.l       #RAMBASE+RAMVALID,D0    ;load RAMBASE + valid bit into D0.
movec.l      D0, RAMBAR           ;load RAMBAR and enable SRAM
```

The following loop initializes the entire SRAM to zero

```
lea.l        RAMBASE,A0           ;load pointer to SRAM
move.l       #16384,D0            ;load loop counter into D0

SRAM_INIT_LOOP:
clr.l        (A0)+                ;clear 4 bytes of SRAM
subq.l       #1,D0                ;decrement loop counter
bne.b        SRAM_INIT_LOOP       ;if done, then exit; else continue looping
```



## 6.2.4 Power Management

As noted previously, depending on the configuration defined by the RAMBAR, instruction fetch and operand read accesses may be sent to the SRAM and cache simultaneously. If the access is mapped to the SRAM module, it sources the read data and the unified cache access is discarded. If the SRAM is used only for data operands, setting the ASn bits associated with instruction fetches can decrease power dissipation. Additionally, if the SRAM contains only instructions, masking operand accesses can reduce power dissipation. [Table 6-2](#) shows some examples of typical RAMBAR settings.

**Table 6-2. Typical RAMBAR Setting Examples**

Data Contained in SRAM	RAMBAR[7:0]
Instruction Only	0x2B
Data Only	0x35
Both Instructions And Data	0x21



# Chapter 7

## Clock Module

### 7.1 Introduction

The clock module allows the MCF5275 to be configured for one of several clocking methods. Clocking modes include internal frequency modulated phase-locked loop (PLL) clocking with either an external clock reference or an external crystal reference supported by an internal crystal amplifier. The PLL can also be disabled and an external oscillator can be used to clock the device directly. The clock module contains:

- Crystal amplifier and oscillator (OSC)
- Frequency Modulated Phase-locked loop (PLL)
- Reduced frequency divider (RFD)
- Status and control registers
- Control logic

#### NOTE

Throughout this manual,  $f_{\text{sys}}$  refers to the core frequency and  $f_{\text{sys}/2}$  refers to the internal bus frequency.

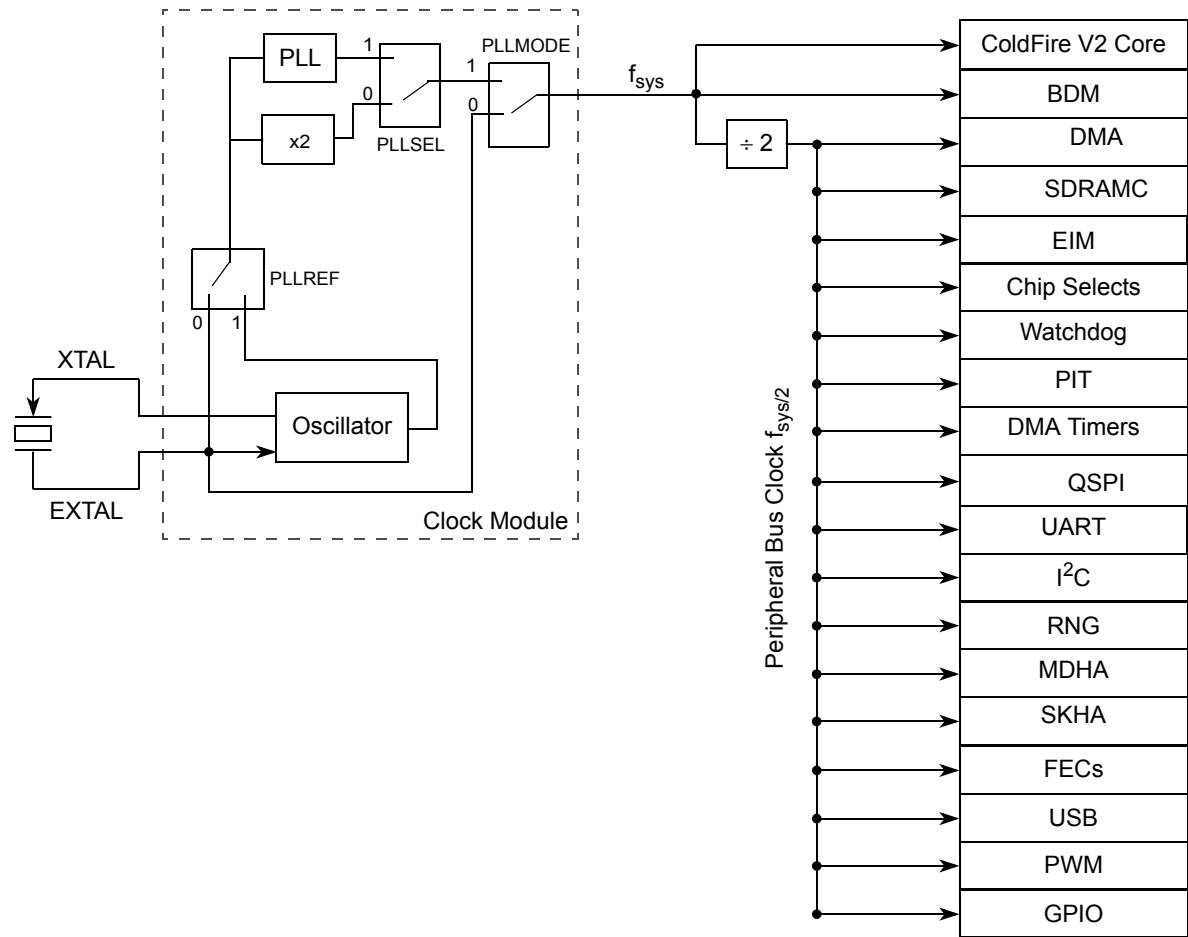
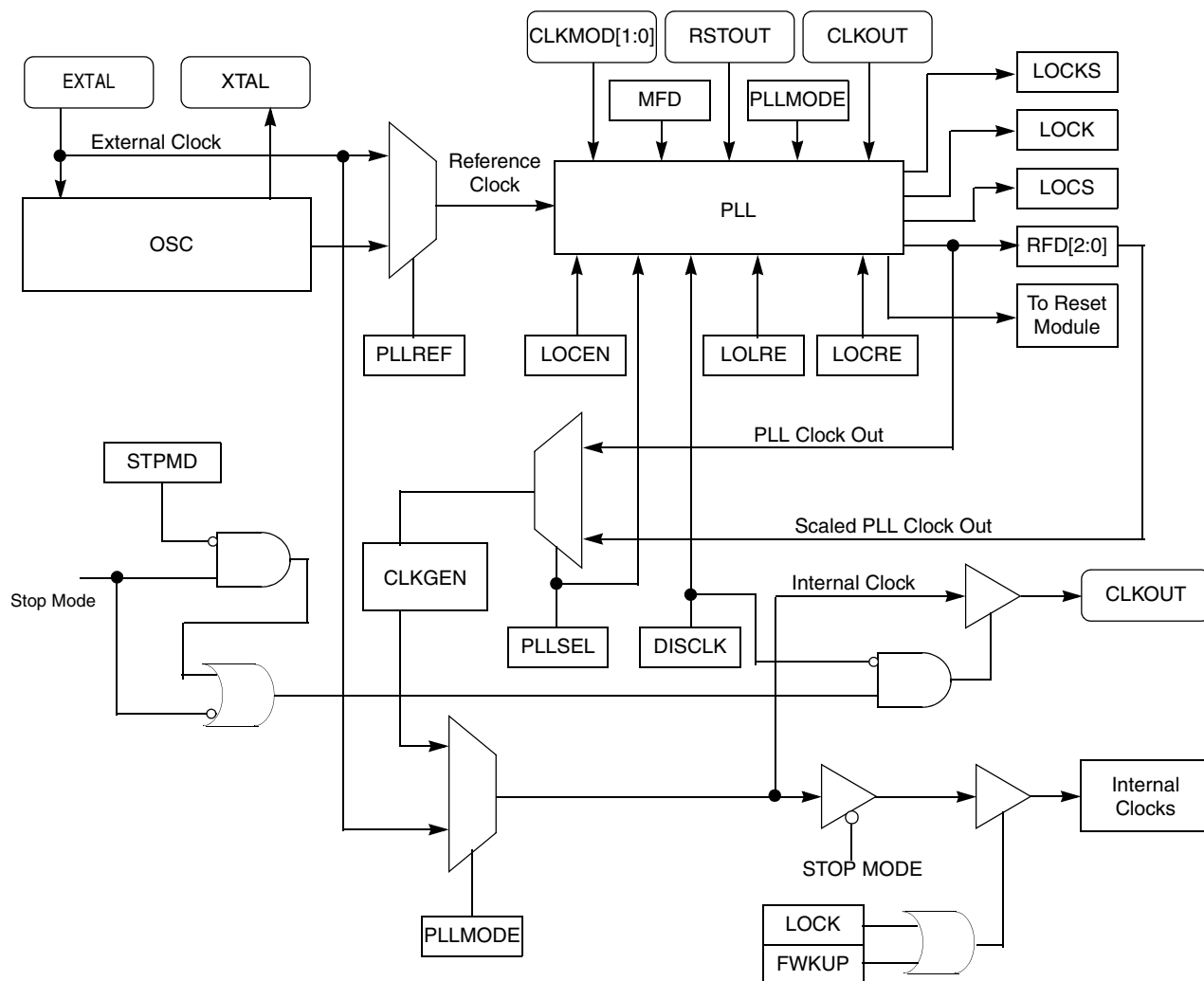


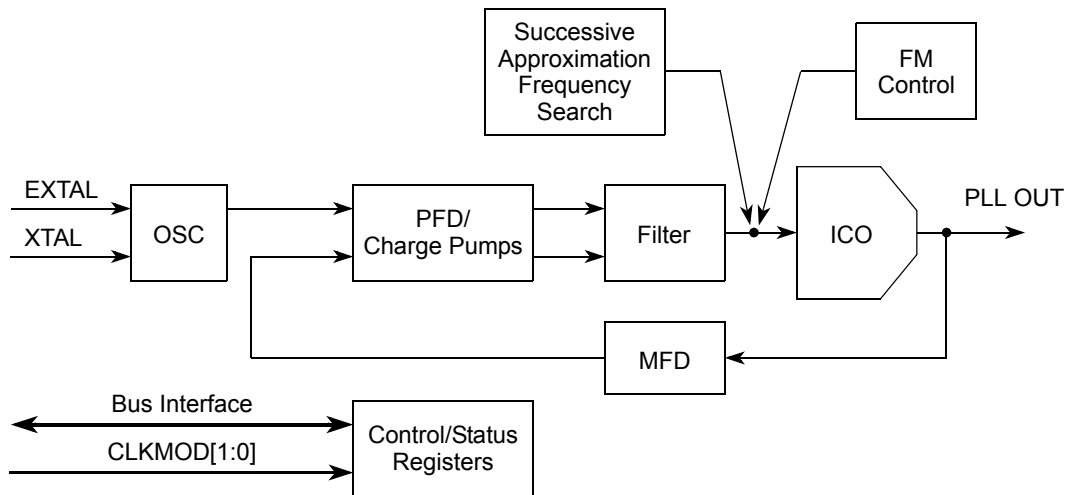
Figure 7-1. MCF5275 Clock Connections

### 7.1.1 Block Diagram

Figure 7-2 shows a block diagram of the entire clock module. The PLL block in this diagram is expanded in detail in Figure 7-3.



**Figure 7-2. Clock Module Block Diagram**



**Figure 7-3. PLL Block Diagram**

## 7.1.2 Features

Features of the clock module include:

- 8- to 25-MHz reference crystal oscillator
- Current controlled oscillator range from 50 MHz to 166 MHz
- Reduced frequency divider for reduced frequency operation without forcing the PLL to re-lock
- Programmable frequency modulation
- Support for low-power modes
- Self-clocked mode operation
- Separate clock out signal

## 7.1.3 Modes of Operation

The PLL operational mode must be configured during reset. The CLKMOD[1:0] package pins must be driven to the appropriate state for the desired mode from the time RSTOUT asserts until it negates. Refer to [Table 7-3](#) for valid states of CLKMOD[1:0]. If CLKMOD[1:0] are not asserted during reset, the PLL will not default to any mode (so these pins must be hard-tied to power or ground for desired mode).

The clock module can be operated in normal PLL mode with crystal reference, normal PLL mode with external reference, 1:1 PLL mode, or external clock mode.

### 7.1.3.1 Normal PLL Mode with Crystal Reference

In normal mode with a crystal reference, the PLL receives an input clock frequency from the crystal oscillator circuit and multiplies the frequency to create the PLL output clock. It can synthesize frequencies ranging from 4x to 18x the reference frequency and has a post divider capable of reducing this synthesized frequency without disturbing the PLL. The user must supply a crystal oscillator that is within the appropriate input frequency range, the crystal manufacture's recommended external support circuitry, and short signal route from the device to the crystal. In normal mode, the PLL can generate a frequency modulated clock or a non-modulated clock (locked on a single frequency). The modulation rate, modulation depth, output clock divide ratio (RFD), and whether the PLL is modulating or not can be programmed by writing to the PLL registers through the bus interface.

### 7.1.3.2 Normal PLL Mode with External Reference

Same as [Section 7.1.3.1, "Normal PLL Mode with Crystal Reference,"](#) except EXTAL is driven by an external clock generator rather than a crystal oscillator. However, the input frequency range is the same as the crystal reference. To enter normal mode with external clock Generator reference, the PLL configuration must be set by following the procedure outlined in [Section 7.4.3, "System Clock Generation."](#)

### 7.1.3.3 1:1 PLL Mode

When 1:1 PLL mode is selected, the PLL synthesizes a core clock frequency equal to two times the input reference frequency ( $f_{\text{sys}}=2 \times f_{\text{ref}}$  and  $f_{\text{sys}/2}=f_{\text{ref}}$ ). The post divider is not active and the frequency modulation capability is not available. Further, modulation must not be present on the input reference clock. The input reference frequency is an external clock reference from a master MCU CLKOUT pin or other external clock generator source. To enter 1:1 PLL mode, the PLL must be set by following the procedure outlined in [Section 7.4.3, "System Clock Generation."](#)

#### NOTE

When configured for 1:1 PLL mode, it is imperative that the CLKOUT clock divider not be changed from its reset state of divide-by-2. Increasing or decreasing this divide ratio will produce unpredictable results from the PLL.

### 7.1.3.4 External Clock Mode (Bypass Mode)

During external clock mode, the PLL is completely bypassed and the user must supply an external clock on the EXTAL pin. The external clock is used directly to produce the internal core clocks. Refer to the Hardware Specification document for external clock input requirements. In external clock mode, the analog portion of the PLL is disabled and no clocks are generated at the PLL.

output. Consequently, frequency modulation is not available. To enter external clock mode, the PLL must be set by following the procedure outlined in [Section 7.4.3, “System Clock Generation.”](#)

### NOTE

XTAL must be tied low in external clock mode when reset is asserted.  
If it is not, clocks could be suspended indefinitely.

## 7.1.3.5 Low-power Mode Operation

This subsection describes the operation of the clock module in low-power and halted modes of operation. Low-power modes are described in [Chapter 8, “Power Management.”](#) [Table 7-1](#) shows the clock module operation in low-power modes.

**Table 7-1. Clock Module Operation in Low-power Modes**

Low-power Mode	Clock Operation	Mode Exit
Wait	Clocks sent to peripheral modules only	Exit not caused by clock module, but normal clocking resumes upon mode exit
Doze	Clocks sent to peripheral modules only	Exit not caused by clock module, but normal clocking resumes upon mode exit
Stop	All system clocks disabled, but clock module continues to run	Exit not caused by clock module, but clock sources are re-enabled and normal clocking resumes upon mode exit

In wait and doze modes, the system clocks to the peripherals are enabled, and the clocks to the CPU, and SRAM are stopped. Each module can disable its clock locally at the module level.

During stop mode, the PLL continues to run. The external CLKOUT signal may be enabled or disabled when the device enters stop mode, depending on the LPCR[STPMD] bit settings.

The external CLKOUT output pin may be disabled to lower power consumption via the SYNCR[DISCLK] bit. The external CLKOUT pin function is enabled by default at reset.

## 7.2 External Signal Descriptions

The clock module signals are summarized in [Table 7-2](#) and a brief description follows. For more detailed information, refer to [Chapter 14, “Signal Descriptions.”](#)

**Table 7-2. Signal Properties**

Name	Function
EXTAL	Oscillator or clock input
XTAL	Oscillator output
CLKOUT	Internal bus clock output



**Table 7-2. Signal Properties (Continued)**

Name	Function
CLKMOD[1:0]	Clock mode select inputs
RSTOUT	Reset signal from reset controller

### 7.2.1 EXTAL

This input is driven by an external clock except when used as a connection to the external crystal when using the internal oscillator.

### 7.2.2 XTAL

This output is an internal oscillator connection to the external crystal.

### 7.2.3 CLKOUT

This output reflects the internal bus clock.

### 7.2.4 CLKMOD[1:0]

The clock mode is selected during reset and reflected in the PLLMODE, PLLSEL, and PLLREF bits of the synthesizer status. Once reset is exited, the clock mode cannot be changed.

The clock mode selection during reset configuration is summarized in [Table 7-3](#).

**Table 7-3. Clock Mode Selection**

Signals		Clock Mode
CLKMOD[1]	CLKMOD[0]	
0	0	PLL Bypass Mode (external clock mode)
0	1	1:1 Mode <sup>1</sup>
1	0	Normal mode with external reference
1	1	Normal mode with crystal reference

<sup>1</sup> In 1:1 mode for the MCF5275,  $f_{\text{sys}} = 2 \times f_{\text{ref\_1:1}}$

### 7.2.5 RSTOUT

The RSTOUT pin is asserted by one of the following:

- Internal system reset signal
- FRCRSTOUT bit in the reset control status register (RCR); see [Section 10.3.1, “Reset Control Register \(RCR\).”](#)

## 7.3 Memory Map/Register Definition

The clock module programming model consists of these registers:

- Synthesizer control register (SYNCR), which defines clock operation
- Synthesizer status register (SYNSR), which reflects clock status

**Table 7-4. Clock Module Memory Map**

IPSBAR Offset	Register Name	Access <sup>1</sup>
0x12_0000	Synthesizer Control Register (SYNCR)	S
0x12_0004	Synthesizer Status Register (SYNSR)	S

<sup>1</sup> S = CPU supervisor mode access only.

### 7.3.1 Register Descriptions

This subsection provides a description of the clock module registers.

#### 7.3.1.1 Synthesizer Control Register (SYNCR)

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	—					MFD[2:0]			—		RFD[2:0]			LOCEN	LOLRE	LOCRE
W																
Reset	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	DISCLK	LOLIRQ	LOCIRQ	RATE	DEPTH	EXP[9:0]										
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Address	IPSBAR + 0x0012_0000															

**Figure 7-4. Synthesizer Control Register (SYNCR)**

Table 7-5. SYNCR Field Descriptions

Bits	Name	Description																																																																																													
31–27	Reserved																																																																																														
26–24	MFD[2:0]	<p>Multiplication factor divider. The MFD bits control the value of the divider in the PLL feedback loop. The value specified by the MFD bits establish the multiplication factor applied to the reference frequency. The bit field encoding is shown in row one of the table below.</p> <p>Note: Frequency modulation should be disabled (see DEPTH bits) prior to making a change to the MFD.</p> <p>The following table illustrates the system frequency multiplier of the reference frequency<sup>1</sup> in normal PLL mode.</p> <table><tr><th colspan="2"></th><th colspan="8">MFD[2:0]</th></tr><tr><th colspan="2"></th><th>000<sup>2</sup> (4x)</th><th>001 (6x)<sup>(3)</sup></th><th>010 (8x)</th><th>011 (10x)</th><th>100 (12x)</th><th>101 (14x)</th><th>110 (16x)</th><th>111 (18x)</th></tr><tr><td rowspan="8">RFD[2:0]</td><td>000 (÷ 1)</td><td>4</td><td>6</td><td>8</td><td>10</td><td>12</td><td>14</td><td>16</td><td>18</td></tr><tr><td>001 (÷ 2)</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td></tr><tr><td>010 (÷ 4)<sup>3</sup></td><td>1</td><td>3/2</td><td>2</td><td>5/2</td><td>3</td><td>7/2</td><td>4</td><td>9/2</td></tr><tr><td>011 (÷ 8)</td><td>1/2</td><td>3/4</td><td>1</td><td>5/4</td><td>3/2</td><td>7/4</td><td>2</td><td>9/4</td></tr><tr><td>100 (÷ 16)</td><td>1/4</td><td>3/8</td><td>1/2</td><td>5/8</td><td>3/4</td><td>7/8</td><td>1</td><td>9/8</td></tr><tr><td>101 (÷ 32)</td><td>1/8</td><td>3/16</td><td>1/4</td><td>5/16</td><td>3/8</td><td>7/16</td><td>1/2</td><td>9/16</td></tr><tr><td>110 (÷ 64)</td><td>1/16</td><td>3/32</td><td>1/8</td><td>5/32</td><td>3/16</td><td>7/32</td><td>1/4</td><td>9/32</td></tr><tr><td>111 (÷ 128)</td><td>1/32</td><td>3/64</td><td>1/16</td><td>5/64</td><td>3/32</td><td>7/64</td><td>1/8</td><td>9/64</td></tr></table> <p><sup>1</sup> <math>f_{\text{sys}} = f_{\text{ref}} \times 2(\text{MFD} + 2)/2^{\text{RFD}}</math>, <math>f_{\text{ref}} \times 2(\text{MFD} + 2) \leq 166\text{MHz}</math>, <math>f_{\text{sys}}/2 \leq 83\text{MHz}</math></p> <p><sup>2</sup> MFD = 000 not valid for <math>f_{\text{ref}} &lt; 3 \text{ MHz}</math></p> <p><sup>3</sup> Default value out of reset</p>			MFD[2:0]										000 <sup>2</sup> (4x)	001 (6x) <sup>(3)</sup>	010 (8x)	011 (10x)	100 (12x)	101 (14x)	110 (16x)	111 (18x)	RFD[2:0]	000 (÷ 1)	4	6	8	10	12	14	16	18	001 (÷ 2)	2	3	4	5	6	7	8	9	010 (÷ 4) <sup>3</sup>	1	3/2	2	5/2	3	7/2	4	9/2	011 (÷ 8)	1/2	3/4	1	5/4	3/2	7/4	2	9/4	100 (÷ 16)	1/4	3/8	1/2	5/8	3/4	7/8	1	9/8	101 (÷ 32)	1/8	3/16	1/4	5/16	3/8	7/16	1/2	9/16	110 (÷ 64)	1/16	3/32	1/8	5/32	3/16	7/32	1/4	9/32	111 (÷ 128)	1/32	3/64	1/16	5/64	3/32	7/64	1/8	9/64
		MFD[2:0]																																																																																													
		000 <sup>2</sup> (4x)	001 (6x) <sup>(3)</sup>	010 (8x)	011 (10x)	100 (12x)	101 (14x)	110 (16x)	111 (18x)																																																																																						
RFD[2:0]	000 (÷ 1)	4	6	8	10	12	14	16	18																																																																																						
	001 (÷ 2)	2	3	4	5	6	7	8	9																																																																																						
	010 (÷ 4) <sup>3</sup>	1	3/2	2	5/2	3	7/2	4	9/2																																																																																						
	011 (÷ 8)	1/2	3/4	1	5/4	3/2	7/4	2	9/4																																																																																						
	100 (÷ 16)	1/4	3/8	1/2	5/8	3/4	7/8	1	9/8																																																																																						
	101 (÷ 32)	1/8	3/16	1/4	5/16	3/8	7/16	1/2	9/16																																																																																						
	110 (÷ 64)	1/16	3/32	1/8	5/32	3/16	7/32	1/4	9/32																																																																																						
	111 (÷ 128)	1/32	3/64	1/16	5/64	3/32	7/64	1/8	9/64																																																																																						
23–22	—	Reserved																																																																																													
21–19	RFD	<p>Reduced frequency divider field. The binary value written to RFD[2:0] is the PLL frequency divisor. See table in MFD bit description. Changing RFD[2:0] does not affect the PLL or cause a relock delay. Changes in clock frequency are synchronized to the next falling edge of the current core clock. To avoid surpassing the allowable core operating frequency, write to RFD[2:0] only when the LOCK bit is set.</p> <p><b>Note:</b> In external clock mode, the RFD bits have no affect.</p>																																																																																													
18	LOCEN	<p>Enables the loss-of-clock function. LOCEN does not affect the loss-of-lock function.</p> <p>0 Loss-of-clock function disabled</p> <p>1 Loss-of-clock function enabled</p> <p><b>Note:</b> In external clock mode, the LOCEN bit has no effect.</p>																																																																																													
17	LOLRE	<p>Loss-of-lock reset enable. This bit determines how the integration module handles a loss-of-lock indication. When operation in normal or 1:1 mode, the PLL must be locked before setting the LOLRE bit. Otherwise reset is immediately asserted.</p> <p>0 Ignore loss-of-clock – no reset</p> <p>1 Reset on loss-of-lock</p> <p><b>Note:</b> In external clock mode, the LOLRE bit has no effect.</p>																																																																																													

Table 7-5. SYNCR Field Descriptions (Continued)

Bits	Name	Description
16	LOCRES	Loss-of-clock reset enable. Determines how the system handles a loss-of-clock condition. When the LOCEN bit is clear, LOCRES has no effect. If the LOCS flag in SYNSR indicates a loss-of-clock condition, setting the LOCRES bit causes an immediate reset. To prevent an immediate reset, the LOCRES bit must be cleared before entering stop mode with the PLL disabled. 0 No reset on loss-of-clock 1 Reset on loss-of-clock <b>Note:</b> In external clock mode, the LOCRES bit has no effect.
15	DISCLK	Disable CLKOUT. This bit determines whether CLKOUT is active. When CLKOUT is disabled it is driven low. 0 CLKOUT driven normally 1 CLKOUT driven low (disabled).
14	LOLIRQ	Loss-of-lock interrupt request. This bit determines if a loss-of-lock is ignored or if an interrupt is requested. When operating in normal or 1:1 PLL mode, the PLL must be locked before setting the LOLIRQ bit. Otherwise an interrupt is immediately requested. 0 Ignore loss-of-lock – no interrupt requested 1 Request interrupt on loss-of-lock <b>Note:</b> In external clock mode, the LOLIRQ bit has no effect.
13	LOCIRQ	Loss-of-clock interrupt request. This bit determines if a loss-of-clock is ignored or if an interrupt is requested. LOCIRQ has no effect when LOCEN is cleared. If the LOCF flag in SYNSR indicates a loss-of-clock condition, setting (or having previously set) the LOCIRQ bit causes an interrupt to be immediately requested. 0 Ignore loss-of-clock – no interrupt requested 1 Request interrupt on loss-of-clock <b>Note:</b> In external clock mode, the LOCIRQ bit has no effect.
12	RATE	Modulation rate. This bit controls the rate of frequency modulation applied to the core frequency. Changing the rate by writing to the RATE bit will initiate the FM calibration sequence. 0 $F_m = F_{ref} / 80$ 1 $F_m = F_{ref} / 40$ <b>Note:</b> Frequency modulation should be disabled prior to making a change to the RATE.
11–10	DEPTH	Frequency modulation depth and enable. This bit field controls depth and enables the frequency modulation. When set to a value other than 0x0, the frequency modulation is automatically enabled. 00 Modulation Depth (% of $f_{sys/2}$ ) = 0 01 Modulation Depth (% of $f_{sys/2}$ ) = $1.0 \pm 0.2$ 10 Modulation Depth (% of $f_{sys/2}$ ) = $2.0 \pm 0.2$ 11 Reserved <b>Note:</b> Frequency modulation should be disabled prior to making a change to the DEPTH.
9–0	EXP	Expected difference value. Writing to this field enables Frequency Modulation (FM). This bit field holds the expected value of the difference between the reference and feedback counters. See <a href="#">Section 7.4.5, “Frequency Modulation Depth Calibration,”</a> for details on how to calculate this value. Entering FM calibration mode requires the user to program the EXP field.

### 7.3.1.2 Synthesizer Status Register (SYNSR)

In the SYNSR, only the LOLF and LOCF flag bits are writeable. Writes to bits other than LOLF and LOCF have no effect.

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	—															
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	—						LOLF	LOC	MODE <sup>1</sup>	PLL SEL <sup>1</sup>	PLL REF <sup>1</sup>	LOCKS <sup>1</sup>	LOCK <sup>2</sup>	LOCF	CAL DONE	CAL PASS
W																
Reset	0	0	0	0	0	0	0	0	See Note 1				See Note 2	0	0	0
Address	IPSBAR + 0x0012_0004															

<sup>1</sup> Reset state determined during reset configuration

<sup>2</sup> Reset state determined during reset

**Figure 7-5. Synthesizer Status Register (SYNSR)**

**Table 7-6. SYNSR Field Descriptions**

Bits	Name	Description
31–10	—	Reserved
9	LOLF	<p>Loss-of-lock flag. This bit provides the interrupt request flag. To clear the flag, write a 1 to this bit. Writing 0 has no effect.</p> <p>This flag will not be set and an interrupt will not be requested under any one of the following conditions: (1) loss-of-lock caused by system reset, (2) writing to the SYNSR to modify the MFD bits, or (3) enabling frequency modulation. The only way to clear this flag is to reset the part or write a 1 to this bit.</p> <p>0 Interrupt service not requested 1 Interrupt service requested</p>
8	LOC	<p>Loss-of-clock status. This bit is an indication of whether a loss-of-clock condition is present when operating in normal and 1:1 PLL modes. 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 which sets this bit and the clocks later return to normal, this bit will be cleared.</p> <p>A loss-of-clock condition can only be detected if LOCEN = 1. See <a href="#">Section 7.4.2, “Clock Operation During Reset.”</a></p> <p>Note: LOC is always cleared in external clock mode.</p> <p>0 Clocks are operating normally 1 Clocks are not operating normally.</p>

**Table 7-6. SYNSR Field Descriptions (Continued)**

Bits	Name	Description
7	PLLMODE	Clock mode. This bit is determined at reset and indicates which clock mode the system is utilizing (see <a href="#">Table 7-7</a> ). See <a href="#">Section 7.4.3, "System Clock Generation,"</a> for details on how to configure the system clock mode during reset. 0 External clock mode 1 PLL clock mode
6	PLLSEL	PLL mode select. This bit is determined at reset and indicates which mode the PLL operates in. PLLSEL is cleared in 1:1 PLL mode and external clock mode. See <a href="#">Chapter 10, "Reset Controller Module,"</a> for details on how to configure the system clock mode during reset. 1 Normal PLL mode (see <a href="#">Table 7-7</a> ) 0 1:1 PLL mode
5	PLLREF	PLL clock reference source. Configured at reset and reflects the PLL reference source in normal PLL mode as shown in <a href="#">Table 7-7</a> . 1 Crystal clock reference 0 External clock reference
4	LOCKS	Sticky indication of PLL lock status. The lock detect function sets the LOCKS bit when the PLL achieves lock after: <ul style="list-style-type: none"> <li>• A system reset</li> <li>• A write to SYNCR that changes the MFD[2:0] bits</li> <li>• Frequency modulation is enabled</li> </ul> When the PLL loses lock, LOCKS is cleared. When the PLL relocks, LOCKS remains cleared until one of the three listed events occurs. Furthermore, reading the LOCKS bit at the same time that the PLL loses lock does not reflect the current loss-of-lock condition. In external clock mode, LOCKS remains cleared after reset. In normal PLL mode and 1:1 PLL mode, LOCKS is set after reset. 0 PLL loss-of-lock since last system reset or MFD change or currently not locked due to exit from STOP with FWKUP set 1 No unintentional PLL loss-of-lock since last system reset or MFD change
3	LOCK	PLL lock status bit. Set when the PLL is locked. PLL lock occurs when the synthesized frequency is within approximately 0.75 percent of the programmed frequency. The PLL loses lock when a frequency deviation of greater than approximately 1.5 percent occurs. Reading the LOCK bit at the same time that the PLL loses lock or acquires lock does not reflect the current condition of the PLL. The power-on reset circuit uses the LOCK bit as a condition for releasing reset. If operating in external clock mode, LOCK remains cleared after reset. 0 PLL not locked 1 PLL locked
2	LOCF	Loss-of-clock flag. This bit provides the interrupt request flag. Write a 1 to this bit to clear the flag. Writing 0 has no effect. Asserting reset will clear the flag. This flag is sticky in the sense that if clocks return to normal after the flag has been set, the bit will remain set until cleared by either writing 1 or asserting reset. 0 Interrupt service not requested 1 Interrupt service request

**Table 7-6. SYNSR Field Descriptions (Continued)**

Bits	Name	Description									
1	CALDONE	Calibration complete. This bit indicates whether the calibration sequence has been completed since the last time frequency modulation was enabled. If CALDONE = 0, the calibration sequence is either 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									
0	CALPASS	Calibration passed. The CALPASS bit tells whether the calibration routine was successful. <table border="1"> <thead> <tr> <th>CALPASS</th><th>CALDONE</th><th></th></tr> </thead> <tbody> <tr> <td>0</td><td>1</td><td>unsuccessful</td></tr> <tr> <td>1</td><td>1</td><td>successful</td></tr> </tbody> </table> <p>When the calibration routine is initiated, CALPASS is set and remains set until either modulation is disabled (by clearing the DEPTH bits in the SYNCR) or a failure occurs within the frequency modulation calibration sequence. 0 Calibration unsuccessful 1 Calibration successful</p>	CALPASS	CALDONE		0	1	unsuccessful	1	1	successful
CALPASS	CALDONE										
0	1	unsuccessful									
1	1	successful									

**Table 7-7. System Clock Modes**

PLLMODE:PLLSEL:PLLREF	Clock Mode
000	External clock mode (bypass PLL)
100	1:1 PLL mode
110	Normal PLL mode with external clock reference
111	Normal PLL mode with crystal reference

## 7.4 Functional Description

This subsection provides a functional description of the clock module.

### 7.4.1 System Clock Modes

The system clock source is determined during reset (see [Table 7-3](#) and [Table 9-11](#)). The values of the CLKMOD signals are latched during reset and are of no importance after reset is negated. If CLKMOD1 or CLKMOD0 is changed during a reset other than power-on reset, the internal clocks may glitch as the system clock source is changed between external clock mode and PLL clock mode. Whenever CLKMOD1 or CLKMOD0 is changed in reset, an immediate loss-of-lock condition occurs.

[Table 7-8](#) shows the clock-out frequency to clock-in frequency relationships for the possible system clock modes. Refer to [Section 7.1.3, “Modes of Operation,”](#) for details on each mode.

**Table 7-8. Clock Out and Clock In Relationships**

System Clock Mode	PLL Options <sup>1</sup>	Cross-Reference
Normal PLL clock mode	- without frequency modulation enabled: $f_{\text{sys}} = \frac{2f_{\text{ref}} \times (\text{MFD} + 2)}{2^{\text{RFD}}}$ - with frequency modulation: $f_{\text{sys}} = \frac{f_{\text{ref}} \times (\text{MFD} + 2) \pm \Delta F_m}{2^{\text{RFD}}}$	Section 7.1.3.1, “Normal PLL Mode with Crystal Reference” and Section 7.1.3.2, “Normal PLL Mode with External Reference”
1:1 PLL clock mode	$f_{\text{sys}/2} = 2 \times f_{\text{ref\_1:1}}$	Section 7.1.3.3, “1:1 PLL Mode”
External clock mode	$f_{\text{sys}/2} = f_{\text{ref}}$	Section 7.1.3.4, “External Clock Mode (Bypass Mode)”

<sup>1</sup>  $f_{\text{ref}}$  = input reference frequency  
 $f_{\text{sys}/2}$  = CLKOUT frequency  
 MFD ranges from 0 to 7  
 RFD ranges from 0 to 7  
 $f_{\text{ref}}$  in external clock mode must not exceed 166MHz  
 $\Delta F_m = f_{\text{sys}/2} \times k$  where  $k = 2 \pm 0.6\%$ ,  $4 \pm 0.6\%$ , or  $6 \pm 0.6\%$ .

## 7.4.2 Clock Operation During Reset

In external clock mode, the system is static and does not recognize reset until clocks are applied to EXTAL and XTAL.

In PLL mode, the PLL operates in self-clocked mode (SCM) during reset until the input reference clock to the PLL begins operating within the limits given in the electrical specifications.

If a PLL failure causes a reset, the system enters reset using the reference clock. Then the system clock source changes to the PLL operating in SCM. If SCM is not functional, the system becomes static. Alternately, if the LOCEN bit in SYNCR is cleared when the PLL fails, the system becomes static. If external reset is asserted, the system cannot enter reset unless the PLL is capable of operating in SCM.

### 7.4.2.1 Power-On Reset (POR)

When a POR is detected, the PLL registers will be initialized to their default state. The PLL will not begin operating until the VDDPLL POR signal has negated. At this point, the PLL will begin operating in SCM until a valid reference clock becomes present as indicated by the loss-of-clock circuit. Refer to [Section 10.4.1.1, “Power-On Reset,”](#) for more information.



### 7.4.2.2 External Reset

Once POR for both the device and the VDDPLL supplies have negated, the PLL will begin its lock detect algorithm. However, if a valid reference is not present, the PLL will continue to operate in SCM until one is present. The system will not come out of reset until a valid reference is present and the PLL has acquired lock at the default MFD (see [Table 7-5](#) for the default MFD value).

Following the initial lock with the default MFD, the MFD in the SYNCR may be modified for the desired operating frequency. If the PLL is not able to lock due to an MFD and crystal frequency combination that attempts to force the current controlled oscillator (ICO) outside of its operating range, reset will not negate.

Refer to [Section 10.4.1.1, “Power-On Reset,”](#) for more information.

#### NOTE

When running in an unlocked state, the clocks generated by the PLL are not guaranteed to be stable and may exceed the maximum specified frequency of the device. It is always recommended that the RFD be used as described in [Section 7.4.3, “System Clock Generation,”](#) to insulate the system from any potential frequency overshoot of the PLL clocks.

### 7.4.3 System Clock Generation

In normal PLL clock mode, the default core frequency is one and a half times (1.5x) the reference frequency after reset. The RFD[2:0] and MFD[2:0] bits in the SYNCR select the frequency multiplier with default values of RFD = 0b010 ( $\div 4$ ) and MFD = 0b001 ( $\times 6$ ) (see [Table 7-5](#)).

When programming the PLL, do not exceed the maximum system clock frequency listed in the electrical specifications. Use this procedure to accommodate the frequency overshoot that occurs when the MFD bits are changed. If frequency modulation is going to be enabled, the maximum allowable frequency must be reduced by the programmed  $\Delta F_m$ .

1. Determine the appropriate value for the MFD and RFD fields in the SYNCR; remember to include the  $\Delta F_m$  if frequency modulation is enabled. The amount of jitter in the system clocks can be minimized by selecting the maximum MFD factor that can be paired with an RFD factor to provide the required frequency. See [Table 7-5](#).
2. Write a value of RFD factor (from step 1) + 1 to the RFD field of the SYNCR.
3. If frequency modulation is enabled (by writing to the EXP bit field), disable frequency modulation by writing 0x0 to the DEPTH field of the SYNCR.
4. If programming the MFD, write the MFD value from step 1 to the SYNCR. If enabling frequency modulation, skip this step.

5. Monitor the LOCK flag in SYNSR. When the PLL achieves lock, write the RFD value from step 1 to the RFD field of the SYNCR. This changes the system clocks frequency to the required frequency.
6. If frequency modulation was enabled initially, it can be re-enabled following the steps listed in [Section 7.4.4, “Programming the Frequency Modulation.”](#)

### NOTE

Keep the maximum system clock frequency below the limit given in the Electrical Characteristics.

## 7.4.4 Programming the Frequency Modulation

Frequency modulation is useful for spreading the energy over a broader frequency spectrum in order to reduce EMI.

In normal PLL clock mode, the default synthesis mode is without frequency modulation enabled. When frequency modulation (FM) is enabled three parameters must be set to generate the desired level of modulation: the RATE, DEPTH, and EXP bit fields of the SYNCR. RATE and DEPTH determine the modulation rate and the modulation depth. The EXP field controls the FM calibration routine described in [Section 7.4.5, “Frequency Modulation Depth Calibration.”](#) The equation that shows how to obtain the values to be programmed for EXP is as follows (see [Section 7.4.5, “Frequency Modulation Depth Calibration,”](#) for details):

$$\text{EXP} = \frac{(2 \times (\text{MFD} + 2) \times \text{M} \times \text{P})}{100}$$

[Figure 7-6](#) illustrates the affects of the parameters and the modulation waveform built into the modulation hardware. The modulation waveform is always a triangle wave and its shape is not programmable.

Note, the modulation rates given are specific to a reference frequency of 8 MHz.  $F_{\text{mod}} = F_{\text{ref}}/Q$  where  $Q = \{40, 80\}$  giving modulation rates of 200 kHz, and 100 kHz. Therefore, the utilization of a non 8 MHz reference will result in scaled modulation rates.

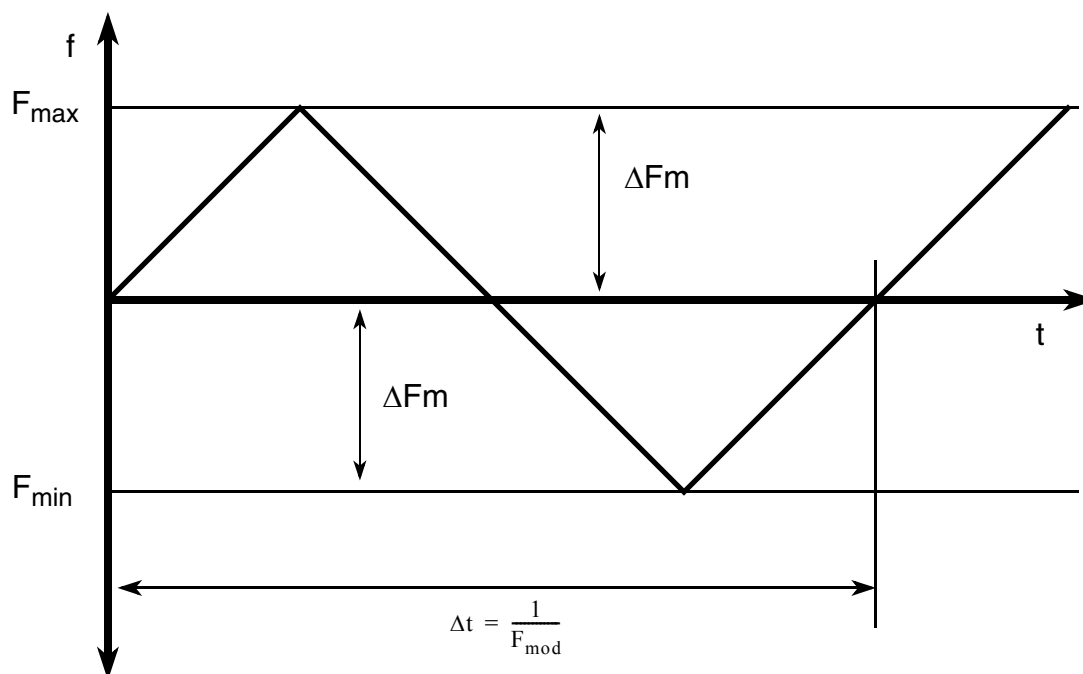
The following steps should be used for proper programming of the frequency modulation mode. These steps ensure proper operation of the calibration routine and prevent frequency overshoot from the sequence.

1. Determine the appropriate value for the EXP field, based upon the selected MFD and desired depth, in the synthesizer control register (SYNCR), as shown in the equation above. Write this value to the EXP field of the SYNCR. The MFD should be programmed to the appropriate value prior to Step 2.
2. Disable modulation by clearing the DEPTH field in the SYNCR.
3. Monitor LOCK bit. Do not proceed until the PLL is locked in non-modulation mode.

4. Write a value of  $RFD = RFD + 1$  to the RFD field of the SYNCR to ensure the maximum system frequency is not exceeded during the calibration routine.
5. Program the desired modulation rates and depths to the RATE and DEPTH fields in the SYNCR. This action initiates the calibration sequence.
6. Allow time for the calibration sequence. Wait for the PLL to lock (the LOCK bit to set in the SYNSR). At this time CALDONE should be asserted. CALPASS will be asserted if the calibration was successful. If not, the calibration can be re-initiated by repeating from step 2. When the PLL achieves lock, write the desired RFD value.

Please note that the frequency modulation system is dependent upon several factors. The accuracies of the VDDPLL/VSSPLL voltage, of the crystal oscillator frequency, and of the manufacturing variation.

For example, if a 5% accurate supply voltage is utilized, then a 5% modulation depth error will result. If the crystal oscillator frequency is skewed from 8MHz, the resulting modulation frequency will be 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%.



$$F_{max} = f_{sys/2} + \{1\%, 2\%\}$$

$$F_{min} = f_{sys/2} - \{1\%, 2\%\}$$

$$F_{mod} = F_{ref}/Q \text{ where } Q = \{40, 80\}$$

**Figure 7-6. Frequency Modulation Waveform**

## 7.4.5 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. Frequency modulation should be disabled prior to making a change to either the MFD, DEPTH, or RATE. Upon enabling frequency modulation a new calibration sequence is performed. A change to MFD, DEPTH, or RATE while in modulation will invalidate calibration results.

Entering the FM calibration mode requires the user to program the EXP field of the SYNCR. Values for EXP can be found using the following equation:

$$\text{EXP} = \frac{(2 \times (\text{MFD} + 2) \times \text{M} \times \text{P})}{100}$$

Example:

For the value of MFD = 4, the number of reference clock cycles to be counted (M) would be 480. Refer to [Figure 7-8](#) for a complete list of values to be used for the variable (M) based on MFD setting. To obtain a percent modulation (P) of 1%, the EXP field would have to be set at:

$$\text{EXP} = (2 \times (4 + 2) \times 480 \times 1) / 100 = 57.6$$

Rounding this value to the closest integer yields the value of 58 that should be entered into the EXP field for this example.

This routine will correct for process variations, but as 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 supply voltages change after the sequence takes place, error incurred will not be corrected. The calibration system reuses the two counters in the lock detect circuit: the reference and feedback counters. The reference counter is still clocked by the reference clock, but the feedback counter is clocked by the ICO clock.

When the calibration routine is initiated (writing to the DEPTH bits), the CALPASS status bit is immediately set and the CALDONE status bit is immediately cleared.

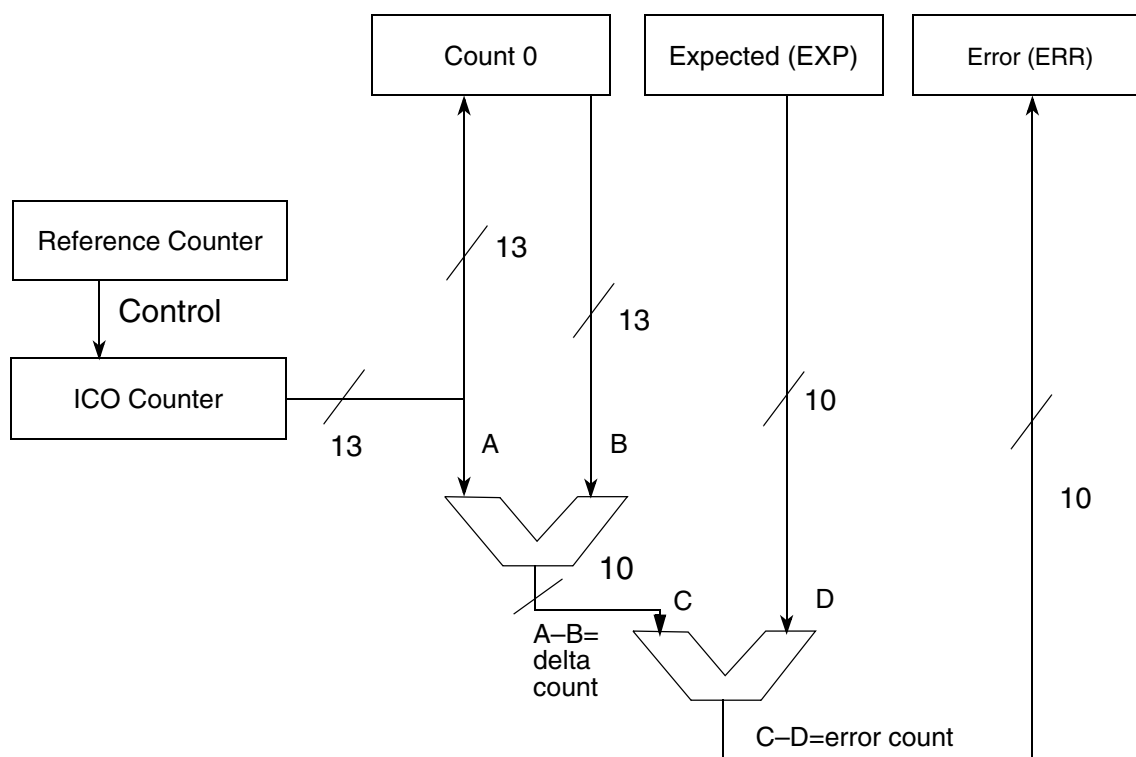
When calibration is induced the ICO is given time to settle. Then both the feedback and reference counters start counting. Full ICO clock cycles are counted by the feedback counter during this time to give the initial center frequency count. When the reference counter has counted to the programmed number 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  $\Delta F_m$ . The ICO is given time to settle. Both counters are reset and restarted. The feedback counter begins to count full ICO clock cycles again to obtain the delta-frequency count. 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 given in the EXP field of the SYNCR register 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 the CALDONE bit of the SYNSR is set. If an error occurs during the calibration routine, then CALPASS is immediately cleared. If the routine completed successfully then CALPASS remains set.

Figure 7-7 shows a block diagram of the calibration circuitry and its associated registers. Figure 7-8 shows a flow chart showing the steps taken by the calibration circuit.



**Figure 7-7. FM Auto-Calibration Data Flow**

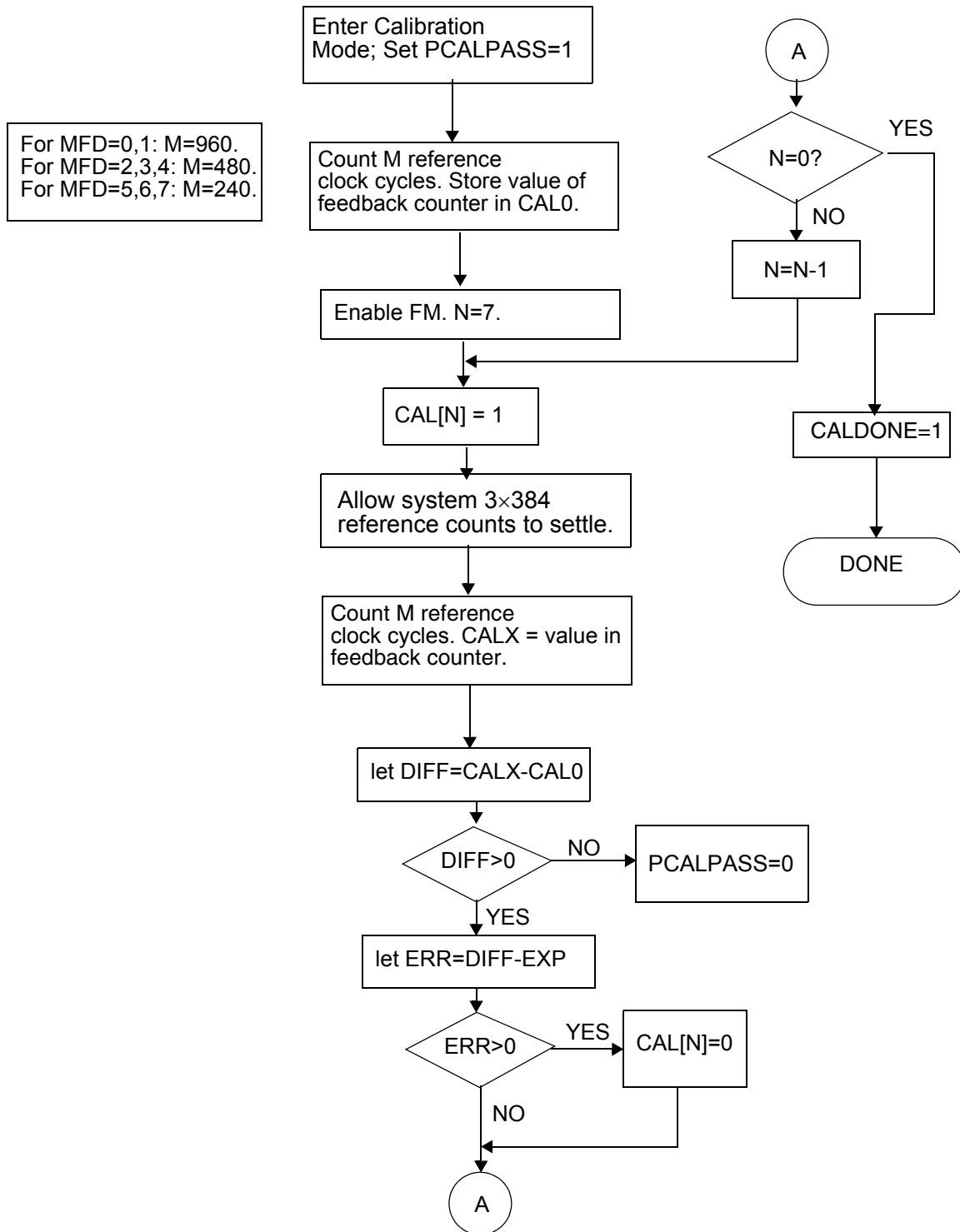
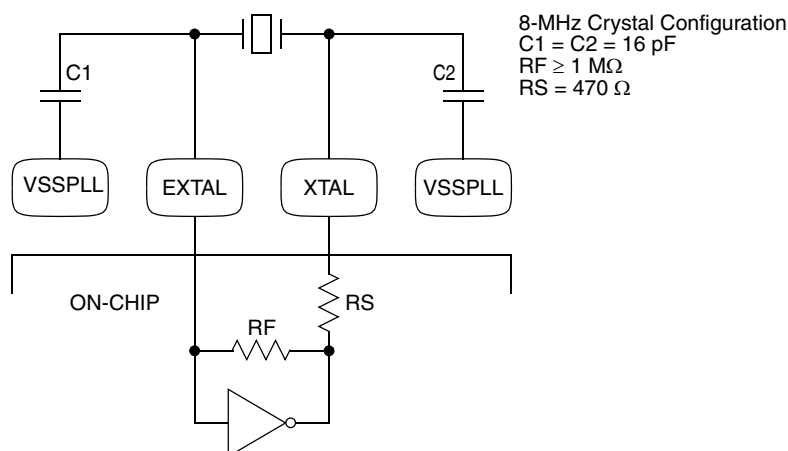


Figure 7-8. FM Auto-calibration Flow Chart

## 7.4.6 PLL Operation

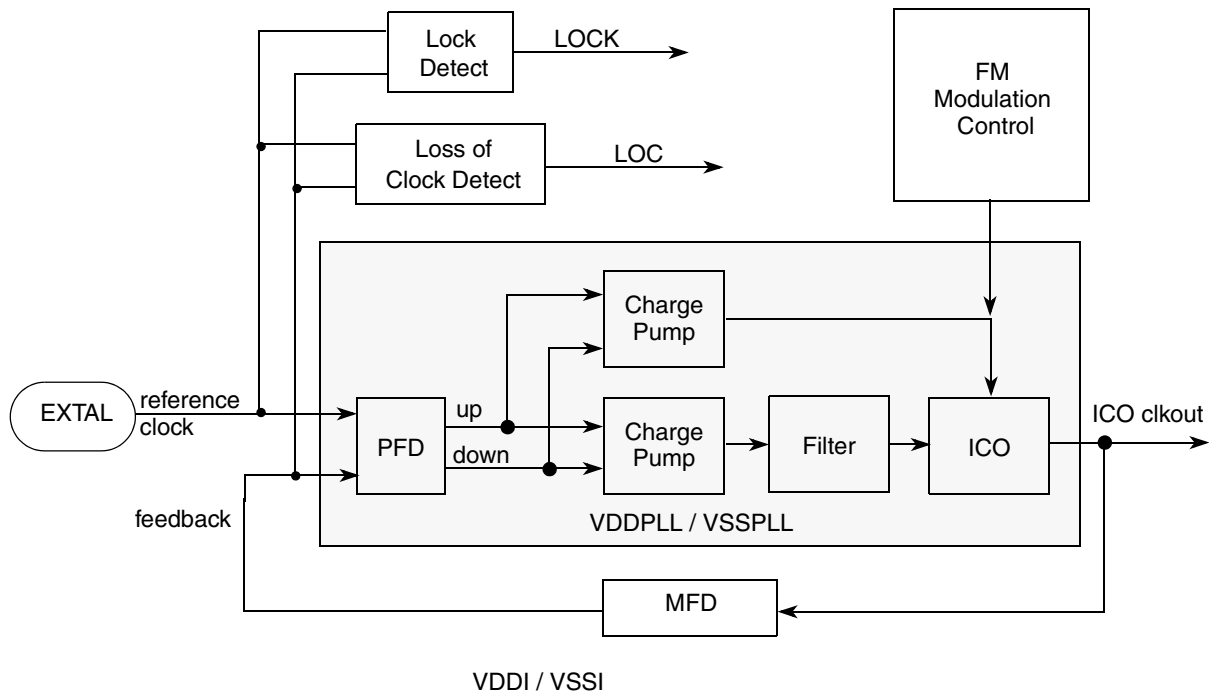
In PLL mode, the PLL synthesizes the system clocks. The PLL can multiply the reference clock frequency by 4x to 18x, provided that the system clock frequency remains within the range listed in the electrical specifications. For example, if the reference frequency is 8 MHz, the PLL can synthesize frequencies of 32 MHz to 144 MHz. In addition, the RFD can reduce the system frequency by dividing the output of the PLL. The RFD is not in the feedback loop of the PLL, so changing the RFD divisor does not affect PLL operation. Finally, the PLL can be frequency modulated to reduce electromagnetic interference often associated with clock circuitry.

Figure 7-9 shows the external support circuitry for the crystal oscillator with example component values. Actual component values depend on crystal specifications.



**Figure 7-9. Crystal Oscillator Example**

The following subsections describe each major block of the PLL. Refer to Figure 7-10 to see how these functional sub-blocks interact.



**Figure 7-10. Frequency Modulated PLL Block Diagram**

#### 7.4.6.1 Phase and Frequency Detector (PFD)

The PFD is a dual-latch phase-frequency detector. It compares both the phase and frequency of the reference and feedback clocks. The reference clock comes from either the crystal oscillator or an external clock source.

The feedback clock comes from one of the following:

- CLKOUT in 1:1 PLL mode
- ICO output divided by two if CLKOUT is disabled in 1:1 PLL mode
- ICO output divided by the MFD in normal PLL mode

When the frequency of the feedback clock equals the frequency of the reference clock, the PLL is frequency-locked. If the falling edge of the feedback clock lags the falling edge of the reference clock, the PFD pulses the UP signal. If the falling edge of the feedback clock leads the falling edge of the reference clock, the PFD pulses the DOWN signal. The width of these pulses relative to the reference clock depends on how much the two clocks lead or lag each other. Once phase lock is achieved, the PFD continues to pulse the UP and DOWN signals for very short durations during each reference clock cycle. These short pulses continually update the PLL and prevent the frequency drift phenomenon known as dead-banding. “Dead-band” is a term used to describe the minimum amount of phase error between the reference and feedback clocks that a phase detector cannot correct.



### 7.4.6.2 Charge Pump/Loop Filter

In 1:1 PLL mode, the charge pump uses a fixed current. In normal mode the current magnitude of the charge pump varies with the MFD as shown in [Table 7-9](#).

**Table 7-9. Charge Pump Current and MFD in Normal Mode Operation**

Charge Pump Current	MFD
1X	$0 \leq \text{MFD} < 2$
2X	$2 \leq \text{MFD} < 6$
4X	$6 \leq \text{MFD}$

The UP and DOWN signals from the PFD control whether the charge pump applies or removes charge, respectively, from the loop filter. The filter is integrated on the chip.

### 7.4.6.3 Current Controlled Oscillator (ICO)

The current into the ICO controls the frequency of the ICO output. The frequency-to-current relationship (ICO gain) is positive.

### 7.4.6.4 Multiplication Factor Divider (MFD)

When the PLL is not in 1:1 PLL mode, the MFD divides the output of the ICO and feeds it back to the PFD. The PFD controls the ICO frequency via the charge pump and loop filter such that the reference and feedback clocks have the same frequency and phase. Thus, the frequency of the input to the MFD, which is also the output of the ICO, is the reference frequency multiplied by the same amount that the MFD divides by. For example, if the MFD divides the ICO frequency by six, the PLL is frequency locked when the ICO frequency is six times the reference frequency. The presence of the MFD in the loop allows the PLL to perform frequency multiplication, or synthesis.

In 1:1 PLL mode, the MFD is bypassed, and the effective multiplication factor is two.

### 7.4.6.5 PLL Lock Detection

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

The lock detect function uses two counters. One is clocked by the reference and the other is clocked by the PLL feedback. When the reference counter has counted N cycles, its count is compared to that of the feedback counter. If the feedback counter has also counted N cycles, the process is repeated for N + K counts. Then, if the two counters still match, the lock criteria is relaxed by one count and the system is notified that the PLL has achieved frequency lock.

After lock is detected, the lock circuit 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 flag 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 tight and relaxed lock criteria prevents the lock detect function from randomly toggling between locked and non-locked status due to phase sensitivities. Figure 7-11 shows the sequence for detecting locked and non-locked conditions.

In external clock mode, the PLL is disabled and cannot lock.

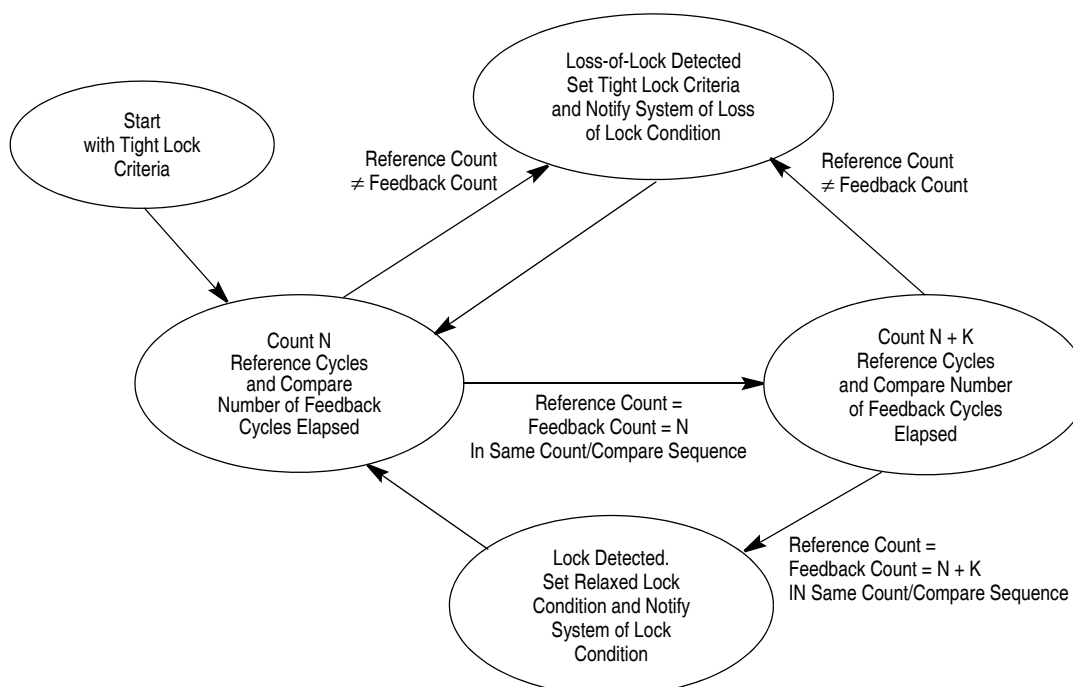


Figure 7-11. Lock Detect Sequence

#### 7.4.6.6 PLL Loss-of-Lock Conditions

Once the PLL acquires lock after reset, the LOCK and LOCKS flags are set. If the MFD is changed, or if an unexpected loss-of-lock condition occurs, the LOCK and LOCKS flags are cleared. While the PLL is in the non-locked condition, the system clocks continue to be sourced from the PLL as the PLL attempts to rellock. Consequently, during the relocking process, the system clocks frequency is not well defined and may exceed the maximum system frequency, violating the system clock timing specifications.

However, once the PLL has relocked, the LOCK flag is set. The LOCKS flag remains cleared if the loss-of-lock is unexpected. The LOCKS flag is set when the loss-of-lock is caused by changing

MFD. If the PLL is intentionally disabled during stop mode, then after exit from stop mode, the LOCKS flag reflects the value prior to entering stop mode once lock is regained.

#### 7.4.6.7 PLL Loss-of-Lock Reset

If the LOLRE bit in the SYNCR is set, a loss-of-lock condition asserts reset. Reset reinitializes the LOCK and LOCKS flags. Therefore, software must read the LOL bit in the reset status register (RSR) to determine if a loss-of-lock caused the reset. See [Section 10.3.2, “Reset Status Register \(RSR\).”](#)

To exit reset in PLL mode, the reference must be present, and the PLL must achieve lock.

In external clock mode, the PLL cannot lock. Therefore, a loss-of-lock condition cannot occur, and the LOLRE bit has no effect.

#### 7.4.6.8 PLL Loss-of-Lock Interrupt Request

The PLL provides the ability to request an interrupt when a loss-of-lock condition occurs by programming the LOLIRQ bit in the SYNCR. An interrupt is requested by the PLL if LOLIRQ is set.

In external clock mode, the PLL cannot lock. Therefore, a loss-of-lock condition cannot occur, and the LOLIRQ bit has no effect.

#### 7.4.6.9 Loss-of-Clock Detection

The LOCEN bit in the SYNCR enables the loss-of-clock detection circuit to monitor the input clocks to the phase and frequency detector (PFD). When either the reference or feedback clock frequency falls below the minimum frequency (see electrical specification for this value), the loss-of-clock circuit sets the sticky LOCF bit, and non-sticky LOC bit, in the SYNSR.

In external clock mode, the loss-of-clock circuit is disabled.

#### 7.4.6.10 Loss-of-Clock Reset

The clock module can assert a reset when a loss-of-clock or loss-of-lock occurs. When a loss-of-clock condition is recognized, reset is asserted if the LOCRE bit in SYNCR is set. The LOCS bit in SYNSR is cleared after reset. Therefore, the LOC bit must be read in the RSR to determine that a loss-of-clock condition occurred. LOCRE has no effect in external clock mode.

To exit reset in PLL mode, the reference must be present, and the PLL must acquire lock.

Reset initializes the clock module registers to a known startup state as described in [Section 7.3, “Memory Map/Register Definition.”](#)

### 7.4.6.11 Loss-of-Clock Interrupt Request

When a loss-of-clock condition is detected, the PLL will request an interrupt if the LOCIRQ bit in the SYNCR is set. The LOCIRQ bit has no affect in external clock mode or if LOCEN is cleared.

### 7.4.6.12 Alternate Clock Selection

Depending on which clock source fails, the loss-of-clock circuit switches the system clocks source to the remaining operational clock. The alternate clock source generates the system clocks until reset is asserted. As [Table 7-10](#) shows, if the reference fails, the PLL goes out of lock and into self-clocked mode (SCM). The PLL remains in SCM until the next reset. When the PLL is operating in SCM, the system frequency depends on the value in the RFD field. The SCM system frequency stated in electrical specifications assumes that the RFD has been programmed to binary 000. If the loss-of-clock condition is due to PLL failure, the PLL reference becomes the system clocks source until the next reset, even if the PLL regains and relocks.

**Table 7-10. Loss-of-Clock Summary**

Clock Mode	System Clock Source Before Failure	Reference Failure Alternate Clock Selected by LOC Circuit <sup>1</sup> Until Reset	PLL Failure Alternate Clock Selected by LOC Circuit Until Reset
PLL	PLL	PLL self-clocked mode	PLL reference
External	External clock	None	NA

<sup>1</sup> The LOC circuit monitors the reference and feedback inputs to the PFD. See [Figure 7-10](#).

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. If the PLL cannot operate in SCM, the system remains static until the next reset. Both the reference and the PLL must be functioning properly to exit reset.

### 7.4.6.13 Loss-of-Clock in Stop Mode

[Table 7-11](#) shows the resulting actions for a loss-of-clock in Stop Mode when the device is being clocked by the various clocking methods.

**Table 7-11. Stop Mode Operation (Sheet 1 of 5)**

MODE In	LOCEN	LOCRE	LOLRE	PLL	OSC	FWKUP	Expected PLL Action at Stop	PLL Action During Stop	MODE Out	LOCKSS	LOCK	LOCS	Comments
EXT	X	X	X	X	X	X	—	—	EXT	0	0	0	
								Lose reference clock	Stuck	—	—	—	

Table 7-11. Stop Mode Operation (Sheet 2 of 5) (Continued)

MODE In	LOCEN	LOCRE	LOLRE	PLL	OSC	FWKUP	Expected PLL Action at Stop	PLL Action During Stop	MODE Out	LOCKSS	LOCK	LOCS	Comments
NRM	0	0	0	Off	Off	0	Lose lock, f.b. clock, reference clock	Regain	NRM	'LK	1	'LC	
								No regain	Stuck	—	—	—	
NRM	X	0	0	Off	Off	1	Lose lock, f.b. clock, reference clock	Regain clocks, but don't regain lock	SCM→unstable NRM	0→'LK	0→1	1→'LC	Block LOCS and LOCKS until clock and lock respectively regain; enter SCM regardless of LOCEN bit until reference regained
								No reference clock regain	SCM→	0→	0→	1→	Block LOCS and LOCKS until clock and lock respectively regain; enter SCM regardless of LOCEN bit
								No f.b. clock regain	Stuck	—	—	—	
NRM	0	0	0	Off	On	0	Lose lock	Regain	NRM	'LK	1	'LC	Block LOCKS from being cleared
								Lose reference clock or no lock regain	Stuck	—	—	—	
								Lose reference clock, regain	NRM	'LK	1	'LC	Block LOCKS from being cleared
NRM	0	0	0	Off	On	1	Lose lock	No lock regain	Unstable NRM	0→'LK	0→1	'LC	Block LOCKS until lock regained
								Lose reference clock or no f.b. clock regain	Stuck	—	—	—	
								Lose reference clock, regain	Unstable NRM	0→'LK	0→1	'LC	LOCS not set because LOCEN = 0

Table 7-11. Stop Mode Operation (Sheet 3 of 5) (Continued)

MODE In	LOCEN	LOCRE	LOLRE	PLL	OSC	FWKUP	Expected PLL Action at Stop	PLL Action During Stop	MODE Out	LOCKSS	LOCK	LOCS	Comments
NRM	0	0	0	On	On	0	—	—	NRM	'LK	1	'LC	
								Lose lock or clock	Stuck	—	—	—	
								Lose lock, regain	NRM	0	1	'LC	
								Lose clock and lock, regain	NRM	0	1	'LC	LOCS not set because LOCEN = 0
NRM	0	0	0	On	On	1	—	—	NRM	'LK	1	'LC	
								Lose lock	Unstable NRM	0	0→1	'LC	
								Lose lock, regain	NRM	0	1	'LC	
								Lose clock	Stuck	—	—	—	
								Lose clock, regain without lock	Unstable NRM	0	0→1	'LC	
								Lose clock, regain with lock	NRM	0	1	'LC	
NRM	X	X	1	Off	X	X	Lose lock, f.b. clock, reference clock	RESET	RESET	—	—	—	Reset immediately
NRM	0	0	1	On	On	X	—	—	NRM	'LK	1	'LC	
								Lose lock or clock	RESET	—	—	—	Reset immediately
NRM	1	0	0	Off	Off	0	Lose lock, f.b. clock, reference clock	Regain	NRM	'LK	1	'LC	REF not entered during stop; SCM entered during stop only during oscillator startup
								No regain	Stuck	—	—	—	
NRM	1	0	0	Off	On	0	Lose lock, f.b. clock	Regain	NRM	'LK	1	'LC	REF mode not entered during stop
								No f.b. clock or lock regain	Stuck	—	—	—	
								Lose reference clock	SCM	0	0	1	Wakeup without lock

Table 7-11. Stop Mode Operation (Sheet 4 of 5) (Continued)

MODE In	LOCEN	LOCRE	LOLRE	PLL	OSC	FWKUP	Expected PLL Action at Stop	PLL Action During Stop	MODE Out	LOCKSS	LOCK	LOCS	Comments
NRM	1	0	0	Off	On	1	Lose lock, f.b. clock	Regain f.b. clock	Unstable NRM	0→'LK	0→1	'LC	REF mode not entered during stop
								No f.b. clock regain	Stuck	—	—	—	
								Lose reference clock	SCM	0	0	1	Wakeup without lock
NRM	1	0	0	On	On	0	—	—	NRM	'LK	1	'LC	
								Lose reference clock	SCM	0	0	1	Wakeup without lock
								Lose f.b. clock	REF	0	X	1	Wakeup without lock
								Lose lock	Stuck	—	—	—	
								Lose lock, regain	NRM	0	1	'LC	
NRM	1	0	0	On	On	1	—	—	NRM	'LK	1	'LC	
								Lose reference clock	SCM	0	0	1	Wakeup without lock
								Lose f.b. clock	REF	0	X	1	Wakeup without lock
								Lose lock	Unstable NRM	0	0→1	'LC	
NRM	1	0	1	On	On	X	—	—	NRM	'LK	1	'LC	
								Lose lock or clock	RESET	—	—	—	Reset immediately
NRM	1	1	X	Off	X	X	Lose lock, f.b. clock, reference clock	RESET	RESET	—	—	—	Reset immediately
NRM	1	1	0	On	On	0	—	—	NRM	'LK	1	'LC	
								Lose clock	RESET	—	—	—	Reset immediately
								Lose lock	Stuck	—	—	—	
								Lose lock, regain	NRM	0	1	'LC	

Table 7-11. Stop Mode Operation (Sheet 5 of 5) (Continued)

MODE In	LOCEN	LOCRE	LOLRE	PLL	OSC	FWKUP	Expected PLL Action at Stop	PLL Action During Stop	MODE Out	LOCKSS	LOCK	LOCS	Comments
NRM	1	1	0	On	On	1	—	—	NRM	'LK	1	'LC	
								Lose clock	RESET	—	—	—	Reset immediately
								Lose lock	Unstable NRM	0	0→1	'LC	
								Lose lock, regain	NRM	0	1	'LC	
NRM	1	1	1	On	On	X	—	—	NRM	'LK	1	'LC	
								Lose clock or lock	RESET	—	—	—	Reset immediately
REF	1	0	0	X	X	X	—	—	REF	0	X	1	
								Lose reference clock	Stuck	—	—	—	
SCM	1	0	0	On	On	0	—	—	SCM	0	0	1	Wakeup without lock
								Lose reference clock	SCM				

**Note:**

PLL = PLL enabled during STOP mode.

OSC = Oscillator enabled during STOP mode.

## MODES

NRM = normal PLL crystal clock reference or normal PLL external reference or PLL 1:1 mode. During PLL 1:1 or normal external reference mode, the oscillator is never enabled. Therefore, during these modes, refer to the OSC = On case regardless of STPMD values.

EXT=external clock mode

REF=PLL reference mode due to losing PLL clock or lock from NRM mode

SCM=PLL self-clocked mode due to losing reference clock from NRM mode

RESET= immediate reset

## LOCKS

'LK= expecting previous value of LOCKS before entering stop

0→'LK= current value is 0 until lock is regained which then will be the previous value before entering stop

0→ = current value is 0 until lock is regained but lock is never expected to regain

## LOCS

'LC=expecting previous value of LOCS before entering stop

1→'LC= current value is 1 until clock is regained which then will be the previous value before entering stop

1→ =current value is 1 until clock is regained but CLK is never expected to regain

## 7.5 Interrupts

Refer to [Section 7.4.6.8, “PLL Loss-of-Lock Interrupt Request,”](#) and [Section 7.4.6.11, “Loss-of-Clock Interrupt Request.”](#)



# Chapter 8

## Power Management

### 8.1 Introduction

This chapter explains the low-power operation of the MCF5275.

#### 8.1.1 Features

The following features support low-power operation.

- Four modes of operation: Run, Wait, Doze, and Stop
- Ability to shut down most peripherals independently
- Ability to shut down the external CLKOUT pin

### 8.2 Memory Map/Register Definition

The PM programming model consists of one register:

- The low-power control register (LPCR) specifies the low-power mode entered when the STOP instruction is issued, and controls clock activity in this low-power mode.

**Table 8-1. Chip Configuration Module Memory Map**

IPSBAR Offset	[31:24]	[23:16]	[15:8]	[7:0]	Access <sup>1</sup>
0x00_0010	Core Reset Status Register (CRSR) <sup>2</sup>	Core Watchdog Control Register (CWCR)	Low-Power Interrupt Control Register (LPICR)	Core Watchdog Service Register (CWSR)	S
0x11_0004	Chip Configuration Register (CCR) <sup>3</sup>		Reserved	Low-Power Control Register (LPCR)	S

<sup>1</sup> S = CPU supervisor mode access only. User mode accesses to supervisor only addresses have no effect and result in a cycle termination transfer error.

<sup>2</sup> The CRSR, CWCR, and CWSR are described in the System Control Module. They are shown here only to warn against accidental writes to these registers when accessing the LPICR.

<sup>3</sup> The CCR is described in the Chip Configuration Module. It is shown here only to warn against accidental writes to this register when accessing the LPCR.

#### 8.2.1 Register Descriptions

The following subsection describes the PM registers.

### 8.2.1.1 Low-Power Interrupt Control Register (LPICR)

Implementation of low-power stop mode and exit from a low-power mode via an interrupt require communication between the CPU and logic associated with the interrupt controller. The LPICR is an 8-bit register that enables entry into low-power stop mode, and includes the setting of the interrupt level needed to exit a low-power mode.

#### NOTE

The setting of the low-power mode select (LPMD) field in the power management module's low-power control register (LPCR) determines which low-power mode the device enters when a STOP instruction is issued.

If this field is set to enter stop mode, then the ENBSTOP bit in the LPICR must also be set.

The following is the sequence of operations needed to enable this functionality:

1. The LPICR is programmed, setting the ENBSTOP bit (if stop mode is the desired low-power mode) and loading the appropriate interrupt priority level.
2. At the appropriate time, the processor executes the privileged STOP instruction. Once the processor has stopped execution, it asserts a specific Processor Status (PST) encoding. Issuing the STOP instruction when the LPICR[ENBSTOP] bit is set causes the SCM to enter stop mode.
3. The entry into a low-power mode is processed by the low-power mode control logic, and the appropriate clocks (usually those related to the high-speed processor core) are disabled.
4. After entering the low-power mode, the interrupt controller enables a combinational logic path which evaluates any unmasked interrupt requests. The device waits for an event to generate an interrupt request with a priority level greater than the value programmed in LPICR[XLPM\_IPL[2:0]].

#### NOTE

Only a fixed (external) interrupt can bring a device out of stop mode. To exit from other low-power modes, such as doze or wait, either fixed or programmable interrupts may be used; however, the module generating the interrupt must be enabled in that particular low-power mode.

5. Once an appropriately high interrupt request level arrives, the interrupt controller signals its presence, and the SCM responds by asserting the request to exit low-power mode.
6. The low-power mode control logic senses the request signal and re-enables the appropriate clocks.
7. With the processor clocks enabled, the core processes the pending interrupt request.

	7	6	5	4	3	2	1	0
R	ENBSTOP		XLPM_IPL[2:0]		0	0	0	0
W								
Reset	0	0	0	0	0	0	0	0
Address	IPSBAR + 0x00_0012							

**Figure 8-1. Low-Power Interrupt Control Register (LPICR)****Table 8-2. LPICR Field Description**

Bits	Name	Description
7	ENBSTOP	Enable low-power stop mode. 0 Low-power stop mode disabled 1 Low-power stop mode enabled. Once the core is stopped and the signal to enter stop mode is asserted, processor clocks can be disabled.
6–4	XLPM_IPL [2:0]	Exit low-power mode interrupt priority level. This field defines the interrupt priority level needed to exit the low-power mode. Refer to <a href="#">Table 8-3</a> .
3–0	—	Reserved, should be cleared.

**Table 8-3. XLPM\_IPL Settings**

XLPM_IPL[2:0]	Interrupts Level Needed to Exit Low-Power Mode
000	Any interrupt request exits low-power mode
001	Interrupt request levels [2-7] exit low-power mode
010	Interrupt request levels [3-7] exit low-power mode
011	Interrupt request levels [4-7] exit low-power mode
100	Interrupt request levels [5-7] exit low-power mode
101	Interrupt request levels [6-7] exit low-power mode
11x	Interrupt request level [7] exits low-power mode

### 8.2.1.2 Low-Power Control Register (LPCR)

The LPCR controls chip operation and module operation during low-power modes.

	7	6	5	4	3	2	1	0
R	LPMD		0	0	STPMD	0	0	0
W								
Reset	0	0	0	0	0	0	0	0
Address	IPSBAR + 0x11_0007							

**Figure 8-2. Low-Power Control Register (LPCR)**

**Table 8-4. LPCR Field Descriptions**

Bits	Name	Description										
7–6	LPMD	<p>Low-power mode select. Used to select the low-power mode the chip enters once the ColdFire CPU executes the STOP instruction. These bits must be written prior to instruction execution for them to take effect. The LPMD[1:0] bits are readable and writable in all modes. Below illustrates the four different power modes that can be configured with the LPMD bit field.</p> <table><tr><th>LPMD[1:0]</th><th>Mode</th></tr><tr><td>11</td><td>STOP</td></tr><tr><td>10</td><td>WAIT</td></tr><tr><td>01</td><td>DOZE</td></tr><tr><td>00</td><td>RUN</td></tr></table> <p><b>Note:</b> If LPCR[LPMD] is cleared, then the MCF5275 will stop executing code upon issue of a STOP instruction. However, no clocks will be disabled.</p>	LPMD[1:0]	Mode	11	STOP	10	WAIT	01	DOZE	00	RUN
LPMD[1:0]	Mode											
11	STOP											
10	WAIT											
01	DOZE											
00	RUN											
5–4	—	Reserved, should be cleared.										
3	STPMD	<p>CLKOUT stop mode. Controls CLKOUT operation during stop mode.</p> <p>0 CLKOUT enabled during stop mode.</p> <p>1 CLKOUT disabled during stop mode.</p>										
2–0	—	Reserved, should be cleared.										

## 8.3 Functional Description

The functions and characteristics of the low-power modes, and how each module is affected by, or affects these modes are discussed in this section.

### 8.3.1 Low-Power Modes

The system enters a low-power mode by executing a STOP instruction. Which mode the device actually enters (either stop, wait, or doze) depends on what is programmed in LPCR[LPMD]. Entry into any of these modes idles the CPU with no cycles active, powers down the system and stops all internal clocks appropriately. During stop mode, the system clock is stopped low.

For entry into stop mode, the LPICR[ENBSTOP] bit must be set before a STOP instruction is issued.

A wakeup event is required to exit a low-power mode and return to run mode. Wakeup events consist of any of these conditions:

- Any type of reset
- Any valid, enabled interrupt request

Exiting from low power mode via an interrupt request requires:

- An interrupt request whose priority is higher than the value programmed in the XLPM\_IPL field of the LPICR.
- An interrupt request whose priority higher than the value programmed in the interrupt priority mask (I) field of the core's status register.
- An interrupt request from a source which is not masked in the interrupt controller's interrupt mask register.
- An interrupt request which has been enabled at the module of the interrupt's origin.

### 8.3.1.1 Run Mode

Run mode is the normal system operating mode. Current consumption in this mode is related directly to the system clock frequency.

### 8.3.1.2 Wait Mode

Wait mode is intended to be used to stop only the CPU and memory clocks until a wakeup event is detected. In this mode, peripherals may be programmed to continue operating and can generate interrupts, which cause the CPU to exit from wait mode.

### 8.3.1.3 Doze Mode

Doze mode affects the CPU in the same manner as wait mode, except that each peripheral defines individual operational characteristics in doze mode. Peripherals which continue to run and have the capability of producing interrupts may cause the CPU to exit the doze mode and return to run mode. Peripherals which are stopped will restart operation on exit from doze mode as defined for each peripheral.

### 8.3.1.4 Stop Mode

Stop mode affects the CPU in the same manner as the wait and doze modes, except that all clocks to the system are stopped and the peripherals cease operation.

Stop mode must be entered in a controlled manner to ensure that any current operation is properly terminated. When exiting stop mode, most peripherals retain their pre-stop status and resume operation.

The following subsections specify the operation of each module while in and when exiting low-power modes.

**NOTE**

Entering stop mode will disable the SDRAMC including the refresh counter. If SDRAM is used, then code is required to insure proper entry and exit from stop mode. See [Section 8.3.2.4, “DDR SDRAM Controller \(SDRAMC\)”](#) for more information.

**8.3.1.5 Peripheral Shut Down**

Most peripherals may be disabled by software in order to cease internal clock generation and remain in a static state. Each peripheral has its own specific disabling sequence (refer to each peripheral description for further details). A peripheral may be disabled at any time and will remain disabled during any low-power mode of operation.

**8.3.2 Peripheral Behavior in Low-Power Modes****8.3.2.1 ColdFire Core**

The ColdFire core is disabled during any low-power mode. No recovery time is required when exiting any low-power mode.

**8.3.2.2 Static Random-Access Memory (SRAM)**

SRAM is disabled during any low-power mode. No recovery time is required when exiting any low-power mode.

**8.3.2.3 System Control Module (SCM)**

The SCM's core watchdog timer can bring the device out of all low-power modes except stop mode. In stop mode, all clocks stop, and the core watchdog does not operate.

When enabled, the core watchdog can bring the device out of low-power mode via a core watchdog interrupt. This system setup must meet the conditions specified in [Section 8.3.1, “Low-Power Modes”](#) for the core watchdog interrupt to bring the part out of low-power mode.

**8.3.2.4 DDR SDRAM Controller (SDRAMC)**

DDR SDRAM Controller operation is unaffected by either the wait or doze modes; however, the SDRAMC is disabled by stop mode. Since all clocks to the SDRAMC are disabled by stop mode, the SDRAMC will not generate refresh cycles.

To prevent loss of data the SDRAMC should be placed in self-refresh mode by clearing `SDRAM_CONTROL[CKE]` and setting `SDRAMC_CONTROL[REF_EN]`. The SDRAM self-refresh mode allows the SDRAM to enter a low-power state where internal refresh operations are used to maintain the integrity of the data stored in the SDRAM.

When stop mode is exited setting the SDRAM\_CONTROL[CKE] bit will cause the SDRAM to exit the self-refresh mode and allow bus cycles to the SDRAM to resume.

### NOTE

The SDRAM is inaccessible while in the self-refresh mode. Therefore, if stop mode is used the vector table and any interrupt handlers that could wake the processor should not be stored in or attempt to access SDRAM.

#### 8.3.2.5 Chip Select Module

In wait and doze modes, the chip select module continues operation but does not generate interrupts; therefore it cannot bring a device out of a low-power mode. This module is stopped in stop mode.

#### 8.3.2.6 DMA Controller (DMA0–DMA3)

In wait and doze modes, the DMA controller is capable of bringing the device out of a low-power mode by generating an interrupt either upon completion of a transfer or upon an error condition. The completion of transfer interrupt is generated when DMA interrupts are enabled by the setting of the DCR[INT] bit, and an interrupt is generated when the DSR[DONE] bit is set. The interrupt upon error condition is generated when the DCR[INT] bit is set, and an interrupt is generated when either the CE, BES or BED bit in the DSR becomes set.

The DMA controller is stopped in stop mode and thus cannot cause an exit from this low-power mode.

#### 8.3.2.7 UART Modules (UART0, UART1, and UART2)

In wait and doze modes, the UART may generate an interrupt to exit the low-power modes.

- Clearing the transmit enable bit (TE) or the receiver enable bit (RE) disables UART functions.
- The UARTs are unaffected by wait mode and may generate an interrupt to exit this mode.

In stop mode, the UARTs stop immediately and freeze their operation, register values, state machines, and external pins. During this mode, the UART clocks are shut down. Coming out of stop mode returns the UARTs to operation from the state prior to the low-power mode entry.

#### 8.3.2.8 I<sup>2</sup>C Module

When the I<sup>2</sup>C Module is enabled by the setting of the I2CR[IEN] bit and when the device is not in stop mode, the I<sup>2</sup>C module is operable and may generate an interrupt to bring the device out of a low-power mode. For an interrupt to occur, the I2CR[IIE] bit must be set to enable interrupts, and the setting of the I2SR[IIF] generates the interrupt signal to the CPU and interrupt controller. The

setting of I2SR[IIF] signifies either the completion of one byte transfer or the reception of a calling address matching its own specified address when in slave receive mode.

In stop mode, the I<sup>2</sup>C Module stops immediately and freezes operation, register values, and external pins. Upon exiting stop mode, the I<sup>2</sup>C resumes operation unless stop mode was exited by reset.

### 8.3.2.9 Queued Serial Peripheral Interface (QSPI)

In wait and doze modes, the queued serial peripheral interface (QSPI) may generate an interrupt to exit the low-power modes.

- Clearing the QSPI enable bit (SPE) disables the QSPI function.
- The QSPI is unaffected by wait mode and may generate an interrupt to exit this mode.

In stop mode, the QSPI stops immediately and freezes operation, register values, state machines, and external pins. During this mode, the QSPI clocks are shut down. Coming out of stop mode returns the QSPI to operation from the state prior to the low-power mode entry.

### 8.3.2.10 DMA Timers (DTIM0–DTIM3)

In wait and doze modes, the DMA timers may generate an interrupt to exit a low-power mode. This interrupt can be generated when the DMA Timer is in either input capture mode or reference compare mode.

In input capture mode, where the capture enable (CE) field of the timer mode register (DTMR) has a non-zero value and the DMA enable (DMAEN) bit of the DMA timer extended mode register (DTXMR) is cleared, an interrupt is issued upon a captured input. In reference compare mode, where the output reference request interrupt enable (ORRI) bit of DTMR is set and the DTXMR[DMAEN] bit is cleared, an interrupt is issued when the timer counter reaches the reference value.

DMA timer operation is disabled in stop mode, but the DMA timer is unaffected by either the wait or doze modes and may generate an interrupt to exit these modes. Upon exiting stop mode, the timer will resume operation unless stop mode was exited by reset.

### 8.3.2.11 Interrupt Controllers (INTC0, INTC1)

The interrupt controller is not affected by any of the low-power modes. All logic between the input sources and generating the interrupt to the processor will be combinational to allow the ability to wake up the CPU processor during low-power stop mode when all system clocks are stopped.

An interrupt request will cause the CPU to exit a low-power mode only if that interrupt's priority level is at or above the level programmed in the interrupt priority mask field of the CPU's status register (SR). The interrupt must also be enabled in the interrupt controller's interrupt mask register as well as at the module from which the interrupt request would originate.



### 8.3.2.12 Fast Ethernet Controller (FEC)

In wait and doze modes, the FEC may generate an interrupt to exit the low-power modes.

- Clearing the ECNTRL[ETHER\_EN] bit disables the FEC function.
- The FEC is unaffected by wait mode and may generate an interrupt to exit this mode.

In stop mode, the FEC stops immediately and freezes operation, register values, state machines, and external pins. During this mode, the FEC clocks are shut down. Coming out of stop mode returns the FEC to operation from the state prior to the low-power mode entry.

### 8.3.2.13 I/O Ports

The I/O ports are unaffected by entry into a low-power mode. These pins may impact low-power current draw if they are configured as outputs and are sourcing current to an external load. If low-power mode is exited by a reset, the state of the I/O pins will revert to their default direction settings.

### 8.3.2.14 Reset Controller

A power-on reset (POR) will always cause a chip reset and exit from any low-power mode.

In wait and doze modes, asserting the external  $\overline{\text{RESET}}$  pin for at least four clocks will cause an external reset that will reset the chip and exit any low-power modes.

In stop mode, the  $\overline{\text{RESET}}$  pin synchronization is disabled and asserting the external  $\overline{\text{RESET}}$  pin will asynchronously generate an internal reset and exit any low-power modes. Registers will lose current values and must be reconfigured from reset state if needed.

If the phase lock loop (PLL) in the clock module is active and if the appropriate (LOCRE, LOLRE) bits in the synthesizer control register are set, then any loss-of-clock or loss-of-lock will reset the chip and exit any low-power modes.

If the watchdog timer is still enabled during wait or doze modes, then a watchdog timer timeout may generate a reset to exit these low-power modes.

When the CPU is inactive, a software reset cannot be generated to exit any low-power mode.

### 8.3.2.15 Chip Configuration Module

The Chip Configuration Module is unaffected by entry into a low-power mode. If low-power mode is exited by a reset, chip configuration may be executed if configured to do so.

### 8.3.2.16 Clock Module

In wait and doze modes, the clocks to the CPU and SRAM will be stopped and the system clocks to the peripherals are enabled. Each module may disable the module clocks locally at the module level. In stop mode, all clocks to the system will be stopped.

During stop mode, the PLL continues to run. The external CLKOUT signal may be enabled or disabled when the device enters stop mode, depending on the LPCR[STPMD] bit settings.

The external CLKOUT output pin may be disabled to lower power consumption via the SYNCR[DISCLK] bit. The external CLKOUT pin function is enabled by default at reset.

### 8.3.2.17 Edge Port

In wait and doze modes, the edge port continues to operate normally and may be configured to generate interrupts (either an edge transition or low level on an external pin) to exit the low-power modes.

In stop mode, there is no system clock available to perform the edge detect function. Thus, only the level detect logic is active (if configured) to allow any low level on the external interrupt pin to generate an interrupt (if enabled) to exit the stop mode.

### 8.3.2.18 Watchdog Timer

In stop mode (or in wait/doze mode, if so programmed), the watchdog ceases operation and freezes at the current value. When exiting these modes, the watchdog resumes operation from the stopped value. It is the responsibility of software to avoid erroneous operation.

When not stopped, the watchdog may generate a reset to exit the low-power modes.

### 8.3.2.19 Programmable Interrupt Timers (PIT0, PIT1, PIT2 and PIT3)

In stop mode (or in doze mode, if so programmed), the programmable interrupt timer (PIT) ceases operation, and freezes at the current value. When exiting these modes, the PIT resumes operation from the stopped value. It is the responsibility of software to avoid erroneous operation.

When not stopped, the PIT may generate an interrupt to exit the low-power modes.

### 8.3.2.20 USB Module

The USB device module is clocked externally, so it operates normally in wait, doze, and stop modes. It is capable of generating an interrupt to wake up the core from the wait and doze modes.

The USB block contains an automatic low power mode in which the module enters suspend mode after a 6.0 ms minimum period of inactivity. When the module receives a wake-up from the USB Host, the transceiver will be re-enabled for normal USB operations.

### 8.3.2.21 PWM Module

The PWM module is user programmable as to how it behaves when the device enters wait mode (PWMCTL[PSWAI]) and doze mode (PWMCTL[PFRZ]). If either of these bits are set the PWM input clock to the prescaler will be disabled during the respective low power mode.

In stop mode the input clock is disabled and PWM generation is halted.

### 8.3.2.22 BDM

Entering halt mode via the BDM port (by asserting the external  $\overline{\text{BKPT}}$  pin) will cause the CPU to exit any low-power mode.

### 8.3.2.23 JTAG

The JTAG (Joint Test Action Group) controller logic is clocked using the TCLK input and is not affected by the system clock. The JTAG cannot generate an event to cause the CPU to exit any low-power mode. Toggling TCLK during any low-power mode will increase the system current consumption.

## 8.3.3 Summary of Peripheral State During Low-Power Modes

The functionality of each of the peripherals and CPU during the various low-power modes is summarized in [Table 8-5](#). The status of each peripheral during a given mode refers to the condition the peripheral automatically assumes when the STOP instruction is executed and the LPCR[LPMD] field is set for the particular low-power mode. Individual peripherals may be disabled by programming its dedicated control bits. The wakeup capability field refers to the ability of an interrupt or reset by that peripheral to force the CPU into run mode.

**Table 8-5. CPU and Peripherals in Low-Power Modes**

Module	Peripheral Status <sup>1</sup> / Wakeup Capability					
	Wait Mode		Doze Mode		Stop Mode	
CPU	Stopped	No	Stopped	No	Stopped	No
SRAM	Stopped	No	Stopped	No	Stopped	No
System Control Module	Enabled	Yes <sup>3</sup>	Enabled	Yes <sup>3</sup>	Stopped	No
SDRAM Controller	Enabled	No	Enabled	No	Stopped	No
Chip Select Module	Enabled	No	Enabled	No	Stopped	No
DMA Controller	Enabled	Yes	Enabled	Yes	Stopped	No
UART0, UART1 and UART2	Enabled	Yes <sup>2</sup>	Enabled	Yes <sup>2</sup>	Stopped	No
I <sup>2</sup> C Module	Enabled	Yes <sup>2</sup>	Enabled	Yes <sup>2</sup>	Stopped	No
QSPI	Enabled	Yes <sup>2</sup>	Enabled	Yes <sup>2</sup>	Stopped	No

**Table 8-5. CPU and Peripherals in Low-Power Modes (Continued)**

Module	Peripheral Status <sup>1</sup> / Wakeup Capability					
	Wait Mode		Doze Mode		Stop Mode	
DMA Timers	Enabled	Yes <sup>2</sup>	Enabled	Yes <sup>2</sup>	Stopped	No
Interrupt controller	Enabled	Yes <sup>2</sup>	Enabled	Yes <sup>2</sup>	Enabled	Yes <sup>2</sup>
Fast Ethernet Controller	Enabled	Yes <sup>2</sup>	Enabled	Yes <sup>2</sup>	Stopped	No
I/O Ports	Enabled	No	Enabled	No	Enabled	No
Reset Controller	Enabled	Yes <sup>3</sup>	Enabled	Yes <sup>3</sup>	Enabled	Yes <sup>3</sup>
Chip Configuration Module	Enabled	No	Enabled	No	Stopped	No
Power Management	Enabled	No	Enabled	No	Stopped	No
Clock Module	Enabled	Yes <sup>2</sup>	Enabled	Yes <sup>2</sup>	Enabled	Yes <sup>2</sup>
Edge port	Enabled	Yes <sup>2</sup>	Enabled	Yes <sup>2</sup>	Stopped	Yes <sup>2</sup>
Watchdog timer	Program	Yes <sup>3</sup>	Program	Yes <sup>3</sup>	Stopped	No
Programmable Interrupt Timers	Enabled	Yes <sup>2</sup>	Program	Yes <sup>2</sup>	Stopped	No
USB	Enabled	Yes <sup>2</sup>	Enabled	Yes <sup>2</sup>	Enabled	No
PWM	Program	No	Program	No	Stopped	No
BDM	Enabled	Yes <sup>4</sup>	Enabled	Yes <sup>4</sup>	Enabled	Yes <sup>4</sup>
JTAG	Enabled	No	Enabled	No	Enabled	No

<sup>1</sup> "Program" Indicates that the peripheral function during the low-power mode is dependent on programmable bits in the peripheral register map.

<sup>2</sup> These modules can generate a interrupt which will exit a low-power mode. The CPU will begin to service the interrupt exception after wakeup.

<sup>3</sup> These modules can generate a reset which will exit any low-power mode.

<sup>4</sup> The BDM logic is clocked by a separate TCLK clock. Entering halt mode via the BDM port exits any low-power mode. Upon exit from halt mode, the previous low-power mode will be re-entered and changes made in halt mode will remain in effect.

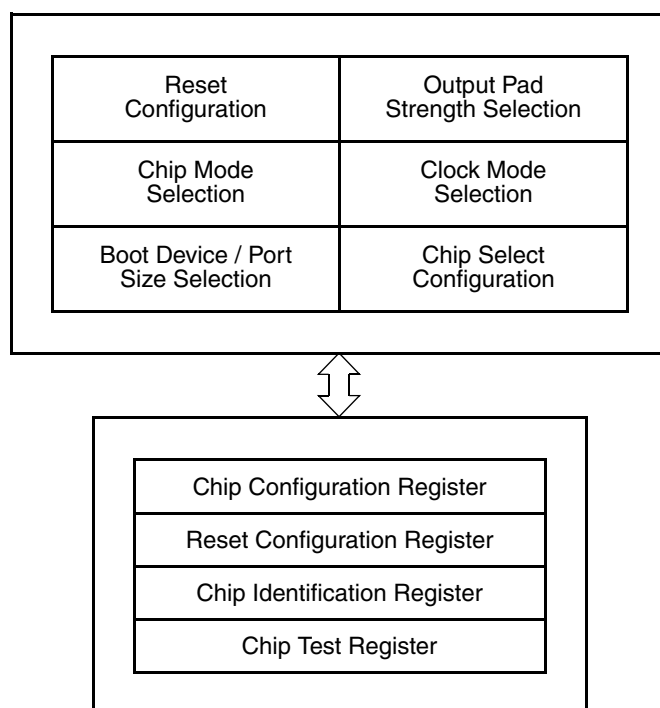
# Chapter 9

## Chip Configuration Module (CCM)

### 9.1 Introduction

The Chip Configuration Module (CCM) controls the chip configuration and mode of operation for the MCF5275.

#### 9.1.1 Block Diagram



**Figure 9-1. Chip Configuration Module Block Diagram**

#### 9.1.2 Features

The CCM performs these operations.

- Selects the chip operating mode
- Selects external clock or phase-lock loop (PLL) mode with internal or external reference
- Selects output pad drive strength
- Selects boot device and data port size
- Selects bus monitor configuration

- Selects low-power configuration
- Selects transfer size function of the external bus
- Selects processor status (PSTAT) and processor debug data (DDATA) functions
- Selects BDM or JTAG mode

### 9.1.3 Modes of Operation

The MCF5275 device only operates in master mode. In master mode, the central processor unit (CPU) can access external memories and peripherals. The external bus consists of a 16-bit data bus and 24 address lines. The available bus control signals include  $\overline{R/W}$ ,  $\overline{TS}$ ,  $\overline{TIP}$ ,  $TSIZ[1:0]$ ,  $\overline{TA}$ ,  $\overline{TEA}$ ,  $\overline{OE}$ , and  $\overline{BS}[1:0]$ . Up to eight chip selects can be programmed to select and control external devices and to provide bus cycle termination..

## 9.2 External Signal Descriptions

Table 9-1 provides an overview of the CCM signals.

**Table 9-1. Signal Properties**

Name	Function	Reset State
$\overline{RCON}$	Reset configuration select	Internal weak pull-up device
CLKMOD[1:0]	Clock mode select <sup>1</sup>	—
D[25:24, 21:19, 16]	Reset configuration override pins	—

<sup>1</sup> Refer to [Chapter 7, “Clock Module”](#) for more information.

### 9.2.1 $\overline{RCON}$

If the external  $\overline{RCON}$  pin is asserted during reset, then various chip functions, including the reset configuration pin functions after reset, are configured according to the levels driven onto the external data pins (see [Section 9.4, “Functional Description”](#)). The internal configuration signals are driven to reflect the levels on the external configuration pins to allow for module configuration.

### 9.2.2 CLKMOD[1:0]

The state of the CLKMOD[1:0] pins during reset determines the clock mode after reset. Refer to [Chapter 7, “Clock Module”](#) for more information.

### 9.2.3 D[25:24, 21:19, 16] (Reset Configuration Override)

If the external  $\overline{RCON}$  pin is asserted during reset, then the states of these data pins during reset determine the chip mode of operation, boot device, clock mode, and certain module configurations after reset.

## 9.3 Memory Map/Register Definition

This subsection provides a description of the memory map and registers.

### 9.3.1 Programming Model

The CCM programming model consists of these registers:

- The chip configuration register (CCR) controls the main chip configuration.
- The reset configuration register (RCON) indicates the default chip configuration.
- The chip identification register (CIR) contains a unique part number.

Some control register bits are implemented as write-once bits. These bits are always readable, but once the bit has been written, additional writes have no effect, except during debug and test operations.

Some write-once bits can be read and written while in debug mode. When debug mode is exited, the chip configuration module resumes operation based on the current register values. If a write to a write-once register bit occurs while in debug mode, the register bit remains writable on exit from debug or test mode. [Table 9-2](#) shows the accessibility of write-once bits.

**Table 9-2. Write-Once Bits Read/Write Accessibility**

Configuration	Read/Write Access
All configurations	Read-always
Debug operation	Write-always
Master mode	Write-once

### 9.3.2 Memory Map

**Table 9-3. Chip Configuration Module Memory Map**

IPSBAR Offset	[31:24]	[23:16]	[15:8]	[7:0]	Access <sup>1</sup>
0x11_0004	Chip Configuration Register (CCR)		Low-Power Control Register (LPCR) <sup>2</sup>		S
0x11_0008	Reset Configuration Register (RCON)		Chip Identification Register (CIR)		S
0x11_000C	Reserved <sup>3</sup>				S
0x11_0010	Unimplemented <sup>4</sup>				—

<sup>1</sup> S = CPU supervisor mode access only. User mode accesses to supervisor only addresses have no effect and result in a cycle termination transfer error.

<sup>2</sup> See [Chapter 8, “Power Management,”](#) for a description of the LPCR. It is shown here only to warn against accidental writes to this register.

<sup>3</sup> Writing to reserved addresses with values other than 0 could put the device in a test mode; reading returns 0s.

<sup>4</sup> Accessing an unimplemented address has no effect and causes a cycle termination transfer error.

**NOTE**

To safeguard against unintentionally activating test logic, write 0x0000 to the above reserved location during initialization (immediately after reset) to lock out test features. Setting any bits in the CCR may lead to unpredictable results.

### 9.3.3 Register Descriptions

The following subsection describes the CCM registers.

#### 9.3.3.1 Chip Configuration Register (CCR)

	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	SZEN	PSTEN	0	BME	BMT		
W																
Reset	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0
Address	IPSBAR + 0x11_0004															

**Figure 9-2. Chip Configuration Register (CCR)**

**Table 9-4. CCR Field Descriptions**

Bits	Name	Description
15–7	—	Reserved, should be cleared.
6	SZEN	TSIZ[1:0] enable. This read/write bit enables the TSIZ[1:0] function of the external pins. 0 TSIZ[1:0] function disabled. DMA Acknowledge function enabled on the TSIZ[1:0] pins. 1 TSIZ[1:0] function enabled. DMA Acknowledge function disabled on the TSIZ[1:0] pins.
5	PSTEN	PST[3:0]/DDATA[3:0] enable. This read/write bit enables the Processor Status (PST) and Debug Data (DDATA)n functions of the external pins. 0 PST/DDATA function disabled. 1 PST/DDATA function enabled.
4	—	Reserved, should be cleared.



**Table 9-4. CCR Field Descriptions (Continued)**

Bits	Name	Description
3	BME	Bus monitor enable. This read/write bit enables the bus monitor to operate during external bus cycles. 0 Bus monitor disabled for external bus cycles. 1 Bus monitor enabled for external bus cycles. <a href="#">Table 9-2</a> shows the read/write accessibility of this write-once bit.
2–0	BMT	Bus monitor timing. This field selects the timeout period (in system clocks) for the bus monitor. 000 65536 001 32768 010 16384 011 8192 100 4096 101 2048 110 1024 111 512 <a href="#">Table 9-2</a> shows the read/write accessibility of this write-once bit.

### 9.3.3.2 Reset Configuration Register (RCON)

At reset, RCON determines the default operation of certain chip functions. All default functions defined by the RCON values can only be overridden during reset configuration (see [Section 9.4.1, “Reset Configuration”](#)) if the external  $\overline{\text{RCON}}$  pin is asserted. RCON is a read-only register.

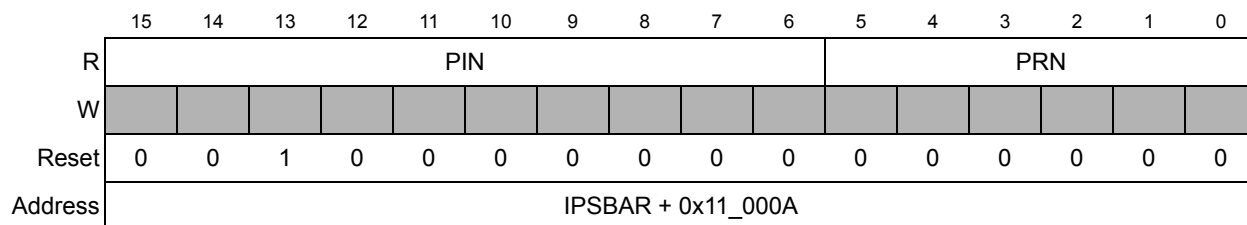
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	RCSC	0	0	RLOAD	BOOTPS	0	0	MODE		
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1
Address	IPSBAR + 0x11_0008															

**Figure 9-3. Reset Configuration Register (RCON)**

**Table 9-5. RCON Field Descriptions**

Bits	Name	Description										
15–10	—	Reserved, should be cleared.										
9–8	RCSC	<div>Chip select configuration. Reflects the default chip select configuration. The default function of the chip select configuration can be overridden during reset configuration.</div> <table><tr><th>RCSC</th><th>Chip Select Configuration</th></tr><tr><td>00<sup>1</sup></td><td>PADDR[7:5] = A[23:21]</td></tr><tr><td>01</td><td>PADDR[7] = <math>\overline{\text{CS}}_6</math> PADDR[6:5] = A[22:21]</td></tr><tr><td>10</td><td>PADDR[7:6] = <math>\overline{\text{CS}}[6:5]</math> PADDR[5] = A[21]</td></tr><tr><td>11</td><td>PADDR[7:5] = <math>\overline{\text{CS}}[6:4]</math></td></tr></table> <div><sup>1</sup> This is the default value used for the MCF5275.</div>	RCSC	Chip Select Configuration	00 <sup>1</sup>	PADDR[7:5] = A[23:21]	01	PADDR[7] = $\overline{\text{CS}}_6$ PADDR[6:5] = A[22:21]	10	PADDR[7:6] = $\overline{\text{CS}}[6:5]$ PADDR[5] = A[21]	11	PADDR[7:5] = $\overline{\text{CS}}[6:4]$
RCSC	Chip Select Configuration											
00 <sup>1</sup>	PADDR[7:5] = A[23:21]											
01	PADDR[7] = $\overline{\text{CS}}_6$ PADDR[6:5] = A[22:21]											
10	PADDR[7:6] = $\overline{\text{CS}}[6:5]$ PADDR[5] = A[21]											
11	PADDR[7:5] = $\overline{\text{CS}}[6:4]$											
7–6	—	Reserved, should be cleared.										
5	RLOAD	<div>Pad driver load. Reflects the default pad driver strength configuration.</div> <div>0 Partial drive strength (This is the default value used for the MCF5275.)</div> <div>1 Full drive strength</div>										
4–3	BOOTPS	<div>Boot port size. Reflects the default selection for the boot port size. The below table shows the different port configurations for BOOTPS. The default function of the boot port size can be overridden during reset configuration.</div> <table><tr><th>BOOTPS[1:0]</th><th>Boot Port Size</th></tr><tr><td>00, 11</td><td>Reserved</td></tr><tr><td>01<sup>1</sup></td><td>External 16 bits</td></tr><tr><td>10</td><td>External 8 bits</td></tr></table> <div><sup>1</sup> This is the default value used for the MCF5275</div>	BOOTPS[1:0]	Boot Port Size	00, 11	Reserved	01 <sup>1</sup>	External 16 bits	10	External 8 bits		
BOOTPS[1:0]	Boot Port Size											
00, 11	Reserved											
01 <sup>1</sup>	External 16 bits											
10	External 8 bits											
2–1	—	Reserved, should be cleared.										
0	MODE	<div>Chip configuration mode. Reflects the default chip configuration mode.</div> <div>0 Reserved</div> <div>1 Master mode (This is the only value available for the MCF5275.)</div> <div>The default mode cannot be overridden during reset configuration.</div>										

### 9.3.3.3 Chip Identification Register (CIR)



**Figure 9-4. Chip Identification Register (CIR)**

**Table 9-6. CIR Field Description**

Bits	Name	Description
15–6	PIN	Part identification number. Contains a unique identification number for the device.
5–0	PRN	Part revision number. This number is increased by one for each new full-layer mask set of this part. The revision numbers are assigned in chronological order.

## 9.4 Functional Description

Six functions are defined within the chip configuration module:

1. Reset configuration
2. Chip mode selection
3. Boot device selection
4. Output pad strength configuration
5. Clock mode selections
6. Chip select configuration

These functions are described here.

### 9.4.1 Reset Configuration

During reset, the pins for the reset override functions are immediately configured to known states. [Table 9-7](#) shows the states of the external pins while in reset.

**Table 9-7. Reset Configuration Pin States During Reset**

Pin	Pin Function <sup>1</sup>	I/O	Output State	Input State
D[25:24, 21:19, 16],	Primary function	Input	—	Must be driven by external logic
RCON	$\overline{\text{RCON}}$ function for all modes <sup>2</sup>	Input	—	Internal weak pull-up device
CLKMOD1, CLKMOD0	Not affected	Input	—	Must be driven by external logic

<sup>1</sup> If the external  $\overline{\text{RCON}}$  pin is not asserted during reset, pin functions are determined by the default operation mode defined in the RCON register. If the external  $\overline{\text{RCON}}$  pin is asserted, pin functions are determined by the override values driven on the external data bus pins.

<sup>2</sup> During reset, the external  $\overline{\text{RCON}}$  pin assumes its  $\overline{\text{RCON}}$  pin function, but this pin changes to the function defined by the chip operation mode immediately after reset. See [Table 9-8](#).

If the  $\overline{\text{RCON}}$  pin is not asserted during reset, the chip configuration and the reset configuration pin functions after reset are determined by RCON or fixed defaults, regardless of the states of the external data pins. The internal configuration signals are driven to levels specified by the RCON register's reset state for default module configuration.

If the  $\overline{\text{RCON}}$  pin is asserted during reset, then various chip functions, including the reset configuration pin functions after reset, are configured according to the levels driven onto the external data pins. (See [Table 9-8](#)) The internal configuration signals are driven to reflect the levels on the external configuration pins to allow for module configuration.

**Table 9-8. Configuration During Reset<sup>1</sup>**

Pin(s) Affected	Default Configuration	Override Pins in Reset <sup>2,3</sup>	Function
D[31:0], $\overline{\text{R}/\overline{\text{W}}}$ , $\overline{\text{TA}}$ , $\overline{\text{TEA}}$ , TSIZ[1:0], $\overline{\text{TS}}$ , $\overline{\text{TIP}}$ , $\overline{\text{OE}}$ , A[23:0], $\overline{\text{BS}}$ [3:0], $\overline{\text{CS}}$ [3:0]	RCON0 = 0	<b>D16</b>	<b>Chip Mode Selected</b>
		1	Master mode <sup>4</sup>
		0	Reserved
CS0	RCON[4:3] = 00	<b>D[20:19]</b>	<b>Boot Device</b>
		00,11	Reserved
		10	External with 8-bit port
		01	External with 16-bit port
All output pins	RCON[5] = 1	<b>D21</b>	<b>Output Pad Drive Strength</b>
		0	Partial strength <sup>4</sup>
		1	Full strength

**Table 9-8. Configuration During Reset<sup>1</sup> (Continued)**

Pin(s) Affected	Default Configuration	Override Pins in Reset <sup>2,3</sup>	Function
Clock mode	No default <sup>5</sup>	<b>CLKMOD1, CLKMOD0</b>	<b>Clock Mode</b>
		00	External clock mode (PLL disabled)
		01	1:1 PLL mode
		10	Normal PLL mode with external clock reference
		11	Normal PLL mode w/crystal reference
A[23:21]/ $\overline{\text{CS}}$ [6:4]	RCON[9:8] = 00	<b>D[25:24]</b>	<b>Chip Select Configuration</b>
		00	PADDR[7:5] = A[23:21] <sup>4</sup>
		10	PADDR[7] = $\overline{\text{CS}}$ PADDR[6:5] = A[22:21]
		01	PADDR[7:6] = $\overline{\text{CS}}$ [6:5] PADDR[5] = A[21]
		11	PADDR[7:5] = $\overline{\text{CS}}$ [6:4]

<sup>1</sup> Modifying the default configurations is possible only if the external  $\overline{\text{RCON}}$  pin is asserted.

<sup>2</sup> The D[31:26, 23:22, 18:17, 15:0] pins do not affect reset configuration.

<sup>3</sup> The external reset override circuitry drives the data bus pins with the override values while  $\overline{\text{RSTOUT}}$  is asserted. It must stop driving the data bus pins within one CLKOUT cycle after  $\overline{\text{RSTOUT}}$  is negated. To prevent contention with the external reset override circuitry, the reset override pins are forced to inputs during reset and do not become outputs until at least one CLKOUT cycle after  $\overline{\text{RSTOUT}}$  is negated. RCON must also be negated within one cycle after  $\overline{\text{RSTOUT}}$  is negated.

<sup>4</sup> Default configuration

<sup>5</sup> There is no default configuration for clock mode selection. The actual values for the CLKMOD pins must always be driven during reset. Once out of reset, the CLKMOD pins have no effect on the clock mode selection.

## 9.4.2 Chip Mode Selection

The chip mode is selected during reset and reflected in the MODE field of the reset configuration register (RCON). See [Section 9.3.3.2, “Reset Configuration Register \(RCON\).”](#) For the MCF5275, there is only one valid chip mode setting.

**Table 9-9. Chip Configuration Mode Selection**

Chip Configuration Mode	RCON[MODE]
Master mode	D16 driven high
Reserved	D16 driven low

During reset, certain module configurations depend on whether emulation mode is active as determined by the state of the internal emulation signal.

### 9.4.3 Boot Device Selection

During reset configuration, the  $\overline{\text{CS0}}$  chip select pin is configured to select an external boot device. In this case, the V (valid) bit in the CSMR0 register is ignored, and  $\overline{\text{CS0}}$  is enabled after reset.  $\overline{\text{CS0}}$  is asserted for the initial boot fetch accessed from address 0x0000\_0000 for the Stack Pointer and address 0x0000\_0004 for the program counter (PC). It is assumed that the reset vector loaded from address 0x0000\_0004 causes the CPU to start executing from external memory space decoded by  $\overline{\text{CS0}}$ .

### 9.4.4 Output Pad Strength Configuration

Output pad strength is determined during reset configuration as shown in [Table 9-10](#).

**Table 9-10. Output Pad Driver Strength Selection<sup>1</sup>**

Optional Pin Function Selection	RCON[RLOAD]
Output pads configured for partial strength	D21 driven low
Output pads configured for full strength	D21 driven high

<sup>1</sup> Modifying the default configurations is possible only if the external RCON pin is asserted low.

### 9.4.5 Clock Mode Selection

The clock mode is selected during reset and reflected in the PLLMODE, PLLSEL, and PLLREF bits of SYNSR. Once reset is exited, the clock mode cannot be changed. [Table 9-11](#) summarizes clock mode selection during reset configuration.

**Table 9-11. Clock Mode Selection<sup>1</sup>**

Clock Mode	CLKMOD[1]	CLKMOD[0]	PLL SYNSR Bits		
			PLLMODE	PLLSEL	PLLREF
External clock mode; PLL disabled	0	0	0	0	0
1:1 PLL mode	0	1	1	0	0
Normal PLL mode; external clock reference	1	0	1	1	0
Normal PLL mode; crystal oscillator reference	1	0	1	1	1

<sup>1</sup> There is no default configuration for clock mode selection. The actual values for the CLKMOD pins must always be driven during reset. Once out of reset, the CLKMOD pins have no effect on the clock mode selection.

### 9.4.6 Chip Select Configuration

The chip select configuration ( $\overline{\text{CS}}[6:4]$ ) is selected during reset and reflected in the RCSC field of the CCR. Once reset is exited, the chip select configuration cannot be changed. [Table 9-8](#) shows the different chip select configurations that can be implemented during reset configuration.

## 9.5 Reset

Reset initializes CCM registers to a known startup state as described in [Section 9.3, “Memory Map/Register Definition.”](#) The CCM controls chip configuration at reset as described in [Section 9.4, “Functional Description.”](#)





# Chapter 10

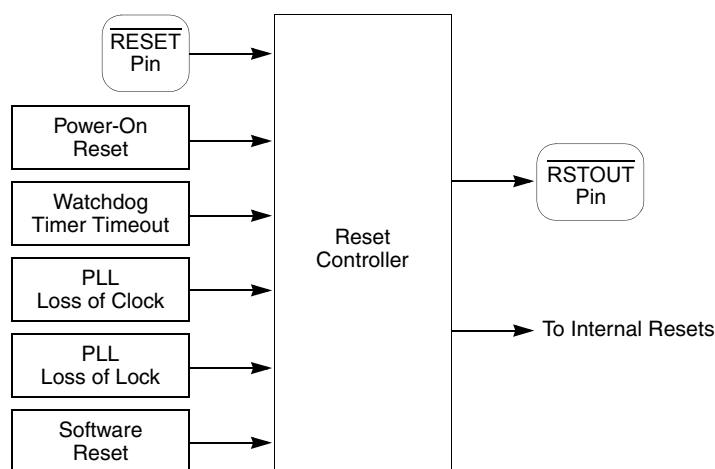
## Reset Controller Module

### 10.1 Introduction

The reset controller is provided to determine the cause of reset, assert the appropriate reset signals to the system, and then to keep a history of what caused the reset.

#### 10.1.1 Block Diagram

Figure 10-1 illustrates the reset controller and is explained in the following sections.



**Figure 10-1. Reset Controller Block Diagram**

#### 10.1.2 Features

Module features include:

- Six sources of reset:
  - External
  - Power-on reset (POR)
  - Watchdog timer
  - Phase locked-loop (PLL) loss of lock
  - PLL loss of clock
  - Software
- Software-assertable  $\overline{\text{RESET}}$  pin independent of chip reset state
- Software-readable status flags indicating the cause of the last reset

## 10.2 External Signal Description

Table 10-1 provides a summary of the reset controller signal properties. The signals are described in the following paragraphs.

**Table 10-1. Reset Controller Signal Properties**

Name	Direction	Input Hysteresis	Input Synchronization
$\overline{\text{RESET}}$	I	Y	Y <sup>1</sup>
$\overline{\text{RSTOUT}}$	O	—	—

<sup>1</sup>  $\overline{\text{RESET}}$  is always synchronized except when in low-power stop mode.

### 10.2.1 $\overline{\text{RESET}}$

Asserting the external  $\overline{\text{RESET}}$  for at least four rising CLKOUT edges causes the external reset request to be recognized and latched.

### 10.2.2 $\overline{\text{RSTOUT}}$

This active-low output signal is driven low when the internal reset controller module resets the chip. When  $\overline{\text{RSTOUT}}$  is active, the user can drive override options on the data bus.

## 10.3 Memory Map/Register Definition

The reset controller programming model consists of these registers:

- Reset control register (RCR), which selects reset controller functions
- Reset status register (RSR), which reflects the state of the last reset source

See Table 10-2 for the memory map and the following paragraphs for a description of the registers.

**Table 10-2. Reset Controller Memory Map**

IPSBAR Offset	[31:24]	[23:16]	[15:8]	[7:0]	Access <sup>1</sup>
0x11_0000	RCR	RSR	Reserved <sup>2</sup>	Reserved <sup>2</sup>	S/U

<sup>1</sup> S/U = supervisor or user mode access.

<sup>2</sup> Writes to reserved address locations have no effect and reads return 0s.

### 10.3.1 Reset Control Register (RCR)

The RCR allows software control for requesting a reset and for independently asserting the external  $\overline{\text{RSTOUT}}$  pin.

	7	6	5	4	3	2	1	0
R	SOFTTRST	FRCRSTOUT	0	0	0	0	0	0
W								
Reset	0	0	0	0	0	0	0	0
Address	IPSBAR + 0x11_0000							

**Figure 10-2. Reset Control Register (RCR)****Table 10-3. RCR Field Descriptions**

Bits	Name	Description
7	SOFTTRST	Allows software to request a reset. The reset caused by setting this bit clears this bit. 1 Software reset request 0 No software reset request
6	FRCRSTOUT	Allows software to assert or negate the external $\overline{\text{RSTOUT}}$ pin. 1 Assert $\overline{\text{RSTOUT}}$ pin 0 Negate $\overline{\text{RSTOUT}}$ pin CAUTION: External logic driving reset configuration data during reset needs to be considered when asserting the $\overline{\text{RSTOUT}}$ pin when setting FRCRSTOUT.
5–0	—	Reserved, should be cleared.

### 10.3.2 Reset Status Register (RSR)

The RSR contains a status bit for every reset source. When reset is entered, the cause of the reset condition is latched along with a value of 0 for the other reset sources that were not pending at the time of the reset condition. These values are then reflected in RSR. One or more status bits may be set at the same time. The cause of any subsequent reset is also recorded in the register, overwriting status from the previous reset condition.

RSR can be read at any time. Writing to RSR has no effect.

	7	6	5	4	3	2	1	0
R	0	0	SOFT	WDR	POR	EXT	LOC	LOL
W								
Reset	0	0	Reset Dependent					
Address	IPSBAR + 0x11_0001							

**Figure 10-3. Reset Status Register (RSR)**

**Table 10-4. RSR Field Descriptions**

Bits	Name	Description
7–6	—	Reserved, should be cleared.
5	SOFT	Software reset flag. Indicates that the last reset was caused by software. 0 Last reset not caused by software 1 Last reset caused by software
4	WDR	Watchdog timer reset flag. Indicates that the last reset was caused by a watchdog timer timeout. 0 Last reset not caused by watchdog timer timeout 1 Last reset caused by watchdog timer timeout
3	POR	Power-on reset flag. Indicates that the last reset was caused by a power-on reset. 0 Last reset not caused by power-on reset 1 Last reset caused by power-on reset
2	EXT	External reset flag. Indicates that the last reset was caused by an external device asserting the external $\overline{\text{RESET}}$ pin. 0 Last reset not caused by external reset 1 Last reset state caused by external reset
1	LOC	Loss-of-clock reset flag. Indicates that the last reset state was caused by a PLL loss of clock. 0 Last reset not caused by loss of clock 1 Last reset caused by loss of clock
0	LOL	Loss-of-lock reset flag. Indicates that the last reset state was caused by a PLL loss of lock. 0 Last reset not caused by loss of lock 1 Last reset caused by a loss of lock

## 10.4 Functional Description

### 10.4.1 Reset Sources

Table 10-5 defines the sources of reset and the signals driven by the reset controller.

**Table 10-5. Reset Source Summary**

Source	Type
Power on	Asynchronous
External $\overline{\text{RESET}}$ pin (not stop mode)	Synchronous
External $\overline{\text{RESET}}$ pin (during stop mode)	Asynchronous
Watchdog timer	Synchronous
Loss of clock	Asynchronous
Loss of lock	Asynchronous
Software	Synchronous

To protect data integrity, a synchronous reset source is not acted upon by the reset control logic until the end of the current bus cycle. Reset is then asserted on the next rising edge of the system clock after the cycle is terminated. Whenever the reset control logic must synchronize reset to the end of the bus cycle, the internal bus monitor is automatically enabled regardless of the BME bit state in the chip configuration register (CCR). Then, if the current bus cycle is not terminated normally the bus monitor terminates the cycle based on the length of time programmed in the BMT field of the CCR.

Internal byte, word, or longword writes are guaranteed to complete without data corruption when a synchronous reset occurs. External writes, including longword writes to 16-bit ports, are also guaranteed to complete.

Asynchronous reset sources usually indicate a catastrophic failure. Therefore, the reset control logic does not wait for the current bus cycle to complete. Reset is asserted immediately to the system.

#### 10.4.1.1 Power-On Reset

At power up, the reset controller asserts  $\overline{\text{RSTOUT}}$ .  $\overline{\text{RSTOUT}}$  continues to be asserted until  $V_{DD}$  has reached a minimum acceptable level and, if PLL clock mode is selected, until the PLL achieves phase lock. Then after approximately another 512 cycles,  $\overline{\text{RSTOUT}}$  is negated and the part begins operation.

#### 10.4.1.2 External Reset

Asserting the external  $\overline{\text{RESET}}$  for at least four rising CLKOUT edges causes the external reset request to be recognized and latched. The bus monitor is enabled and the current bus cycle is completed. The reset controller asserts  $\overline{\text{RSTOUT}}$  for approximately 512 cycles after  $\overline{\text{RESET}}$  is negated and the PLL has acquired lock. The part then exits reset and begins operation.

In low-power stop mode, the system clocks are stopped. Asserting the external  $\overline{\text{RESET}}$  in stop mode causes an external reset to be recognized.

#### 10.4.1.3 Watchdog Timer Reset

A watchdog timer timeout causes timer reset request to be recognized and latched. The bus monitor is enabled and the current bus cycle is completed. If the  $\overline{\text{RESET}}$  is negated and the PLL has acquired lock, the reset controller asserts  $\overline{\text{RSTOUT}}$  for approximately 512 cycles. Then the part exits reset and begins operation.

#### 10.4.1.4 Loss-of-Clock Reset

This reset condition occurs in PLL clock mode when the LOCRE bit in the SYNCR is set and either the PLL reference or the PLL itself fails. The reset controller asserts  $\overline{\text{RSTOUT}}$  for

approximately 512 cycles after the PLL has acquired lock. The part then exits reset and begins operation.

#### 10.4.1.5 Loss-of-Lock Reset

This reset condition occurs in PLL clock mode when the LOLRE bit in the SYNCR is set and the PLL loses lock. The reset controller asserts  $\overline{\text{RSTOUT}}$  for approximately 512 cycles after the PLL has acquired lock. The part then exits reset and resumes operation.

#### 10.4.1.6 Software Reset

A software reset occurs when the SOFTRST bit is set. If the  $\overline{\text{RESET}}$  is negated and the PLL has acquired lock, the reset controller asserts  $\overline{\text{RSTOUT}}$  for approximately 512 cycles. Then the part exits reset and resumes operation.

### 10.4.2 Reset Control Flow

The reset logic control flow is shown in [Figure 10-4](#). In this figure, the control state boxes have been numbered, and these numbers are referred to (within parentheses) in the flow description that follows. All cycle counts given are approximate.

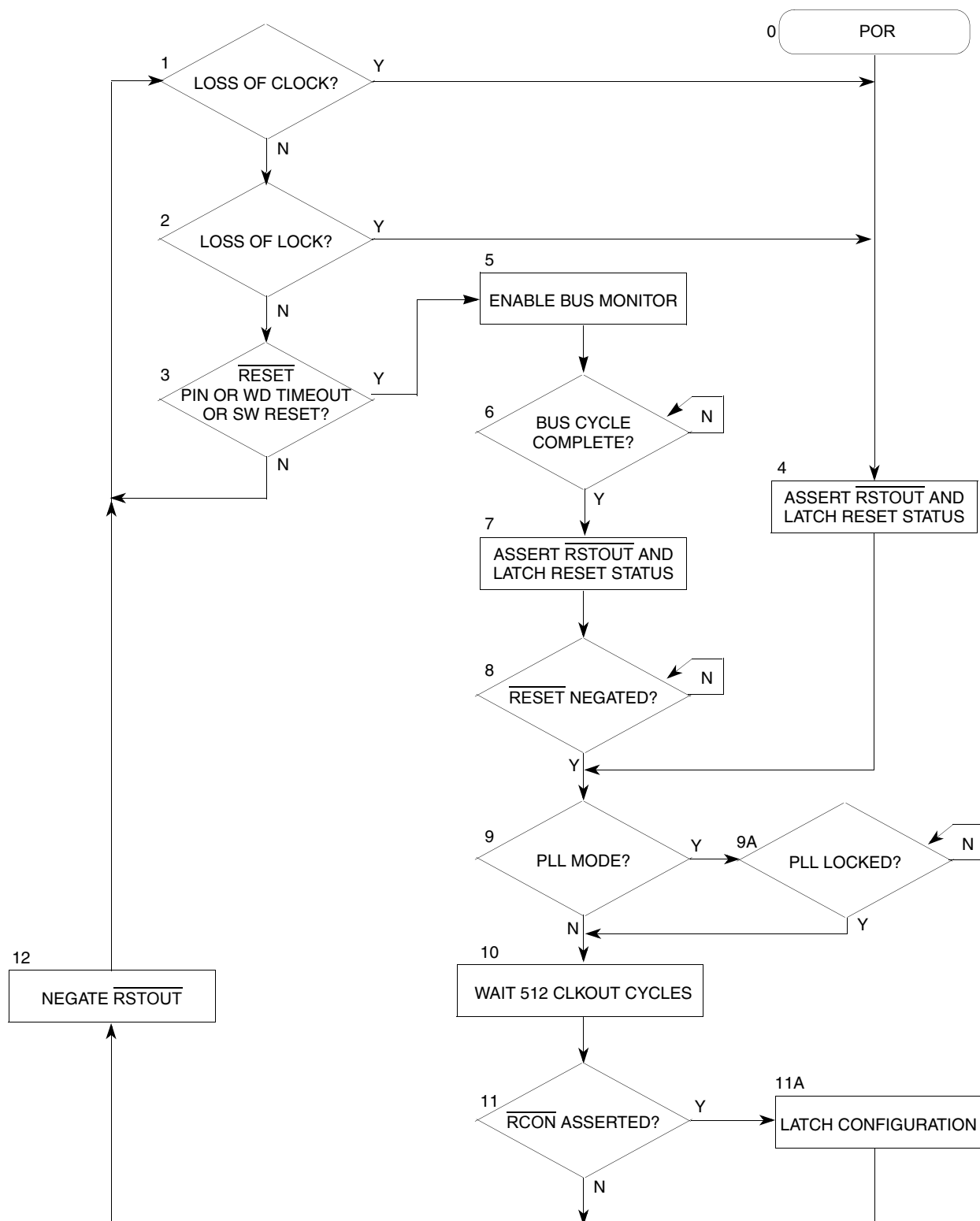


Figure 10-4. Reset Control Flow

### 10.4.2.1 Synchronous Reset Requests

In this discussion, the reference in parentheses refer to the state numbers in [Figure 10-4](#). All cycle counts given are approximate.

If the external  $\overline{\text{RESET}}$  signal is asserted by an external device for at least four rising CLKOUT edges (3), if the watchdog timer times out, or if software requests a reset, the reset control logic latches the reset request internally and enables the bus monitor (5). When the current bus cycle is completed (6),  $\overline{\text{RSTOUT}}$  is asserted (7). The reset control logic waits until the  $\overline{\text{RESET}}$  signal is negated (8) and for the PLL to attain lock (9, 9A) before waiting 512 CLKOUT cycles (1). The reset control logic may latch the configuration according to the  $\overline{\text{RCON}}$  signal level (11, 11A) before negating  $\overline{\text{RSTOUT}}$  (12).

If the external  $\overline{\text{RESET}}$  signal is asserted by an external device for at least four rising CLKOUT edges during the 512 count (10) or during the wait for PLL lock (9A), the reset flow switches to (8) and waits for the  $\overline{\text{RESET}}$  signal to be negated before continuing.

### 10.4.2.2 Internal Reset Request

If reset is asserted by an asynchronous internal reset source, such as loss of clock (1) or loss of lock (2), the reset control logic asserts  $\overline{\text{RSTOUT}}$  (4). The reset control logic waits for the PLL to attain lock (9, 9A) before waiting 512 CLKOUT cycles (1). Then the reset control logic may latch the configuration according to the  $\overline{\text{RCON}}$  pin level (11, 11A) before negating  $\overline{\text{RSTOUT}}$  (12).

If loss of lock occurs during the 512 count (10), the reset flow switches to (9A) and waits for the PLL to lock before continuing.

### 10.4.2.3 Power-On Reset

When the reset sequence is initiated by power-on reset (0), the same reset sequence is followed as for the other asynchronous reset sources.

## 10.4.3 Concurrent Resets

This section describes the concurrent resets. As in the previous discussion references in parentheses refer to the state numbers in [Figure 10-4](#).

### 10.4.3.1 Reset Flow

If a power-on reset is detected during any reset sequence, the reset sequence starts immediately (0).

If the external  $\overline{\text{RESET}}$  pin is asserted for at least four rising CLKOUT edges while waiting for PLL lock or the 512 cycles, the external reset is recognized. Reset processing switches to wait for the external  $\overline{\text{RESET}}$  pin to negate (8).



If a loss-of-clock or loss-of-lock condition is detected while waiting for the current bus cycle to complete (5, 6) for an external reset request, the cycle is terminated. The reset status bits are latched (7) and reset processing waits for the external  $\overline{\text{RESET}}$  pin to negate (8).

If a loss-of-clock or loss-of-lock condition is detected during the 512 cycle wait, the reset sequence continues after a PLL lock (9, 9A).

#### 10.4.3.2 Reset Status Flags

For a POR reset, the POR bit in the RSR are set, and the SOFT, WDR, EXT, LOC, and LOL bits are cleared even if another type of reset condition is detected during the reset sequence for the POR.

If a loss-of-clock or loss-of-lock condition is detected while waiting for the current bus cycle to complete (5, 6) for an external reset request, the EXT, SOFT, and/or WDR bits along with the LOC and/or LOL bits are set.

If the RSR bits are latched (7) during the EXT, SOFT, and/or WDR reset sequence with no other reset conditions detected, only the EXT, SOFT, and/or WDR bits are set.

If the RSR bits are latched (4) during the internal reset sequence with the  $\overline{\text{RESET}}$  pin not asserted and no SOFT or WDR event, then the LOC and/or LOL bits are the only bits set.



# Chapter 11

## System Control Module (SCM)

### 11.1 Introduction

This section details the functionality of the System Control Module (SCM) which provides the programming model for the System Access Control Unit (SACU), the system bus arbiter, a 32-bit core watchdog timer (CWT), and the system control registers and logic. Specifically, the system control includes the internal peripheral system (IPS) base address register (IPSBAR), the processor's dual-port RAM base address register (RAMBAR), and system control registers that include the core watchdog timer control.

#### 11.1.1 Overview

The SCM provides the control and status for a variety of functions including base addressing and address space masking for both the IPS peripherals and resources (IPSBAR) and the ColdFire core memory space (RAMBAR). The MCF5275 CPU core supports one memory bank for the internal SRAM.

The SACU provides the mechanism needed to implement secure bus transactions to the system address space.

The programming model for the system bus arbitration resides in the SCM. The SCM sources the necessary control signals to the arbiter for bus master management.

The CWT provides a means of preventing system lockup due to uncontrolled software loops via a special software service sequence. If periodic software servicing action does not occur, the CWT times out with a programmed response (interrupt) to allow recovery or corrective action to be taken.

#### NOTE

The core watchdog timer is available to provide compatibility with the watchdog timer implemented on previous ColdFire devices. However, there is a second watchdog timer available on the MCF5275 that has new features. See [Chapter 21, “Watchdog Timer Module”](#) for more information. Please note that the core watchdog timer is unable to reset the device. It is only permitted to assert an interrupt. For resetting the device, use the second watchdog timer.

#### 11.1.2 Features

The SCM includes these distinctive features:

- IPS base address register (IPSBAR)
  - Base address location for 1-Gbyte peripheral space
  - User control bits
- Processor-local memory base address register (RAMBAR)
- System control registers
  - Core reset status register (CRSR) indicates type of last reset
  - Core watchdog control register (CWCR) for watchdog timer control
  - Core watchdog service register (CWSR) to service watchdog timer
- System bus master arbitration programming model (MPARK)
- System access control unit (SACU) programming model
  - Master privilege register (MPR)
  - Peripheral access control registers (PACRs)
  - Grouped peripheral access control register (GPACR)

## 11.2 Memory Map/Register Definition

The memory map for the SCM registers is shown in [Table 11-1](#). All the registers in the SCM are memory-mapped as offsets within the 1-Gbyte IPS address space and accesses are controlled to these registers by the control definitions programmed into the SACU.

**Table 11-1. SCM Register Map**

IPSBAR Offset	[31:24]	[23:16]	[15:8]	[7:0]
0x00_0000	IPSBAR			
0x00_0004	—			
0x00_0008	RAMBAR			
0x00_000C	—			
0x00_0010	CRSR	CWCR	LPICR <sup>1</sup>	CWSR
0x00_0014	DMAREQC <sup>2</sup>			
0x00_0018	—			
0x00_001C	MPARK			
0x00_0020	MPR	—		
0x00_0024	PACR0	PACR1	PACR2	PACR3
0x00_0028	PACR4	—	PACR5	PACR6
0x00_002C	PACR7	—	PACR8	—

**Table 11-1. SCM Register Map (Continued)**

IPSBAR Offset	[31:24]	[23:16]	[15:8]	[7:0]
0x00_0030	GPACR	—	—	—
0x00_0034	—	—	—	—
0x00_0038	—	—	—	—
0x00_003C	—	—	—	—

<sup>1</sup> The LPICR register is described in [Chapter 8, “Power Management.”](#)

<sup>2</sup> The DMAREQC register is described in [Chapter 18, “DMA Controller Module.”](#)

## 11.2.1 Register Descriptions

### 11.2.1.1 Internal Peripheral System Base Address Register (IPSBAR)

The IPSBAR specifies the base address for the 1 Gbyte memory space associated with the on-chip peripherals. At reset, the base address is loaded with a default location of 0x4000\_0000 and marked as valid (IPSBAR[V]=1). If desired, the address space associated with the internal modules can be moved by loading a different value into the IPSBAR at a later time.

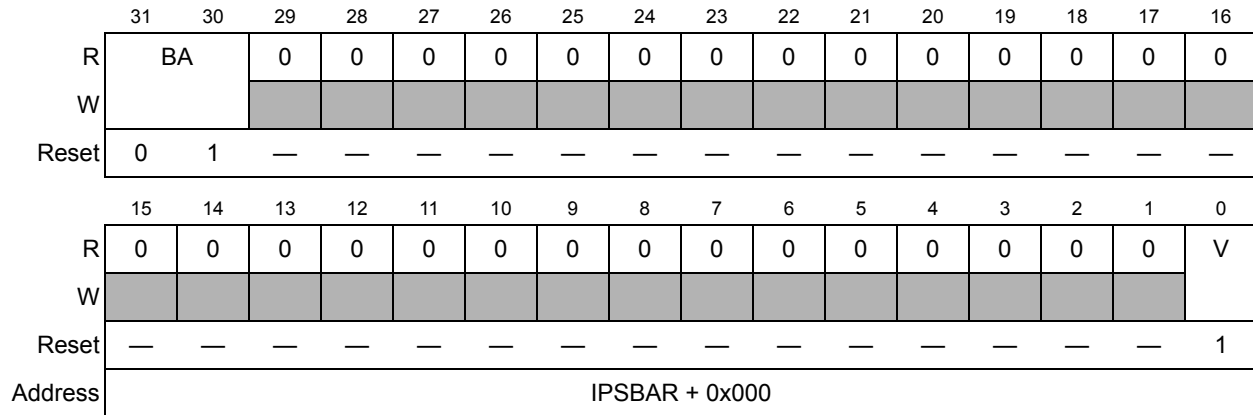
If an address “hits” in overlapping memory regions, the following priority is used to determine what memory is accessed:

1. IPSBAR
2. RAMBAR
3. Cache
4. SDRAM
5. Chip Selects

#### NOTE

This is the list of memory access priorities when viewed from the processor core.

See [Figure 11-1](#) and [Table 11-2](#) for descriptions of the bits in IPSBAR.

**Figure 11-1. IPS Base Address Register (IPSBAR)****Table 11-2. IPSBAR Field Description**

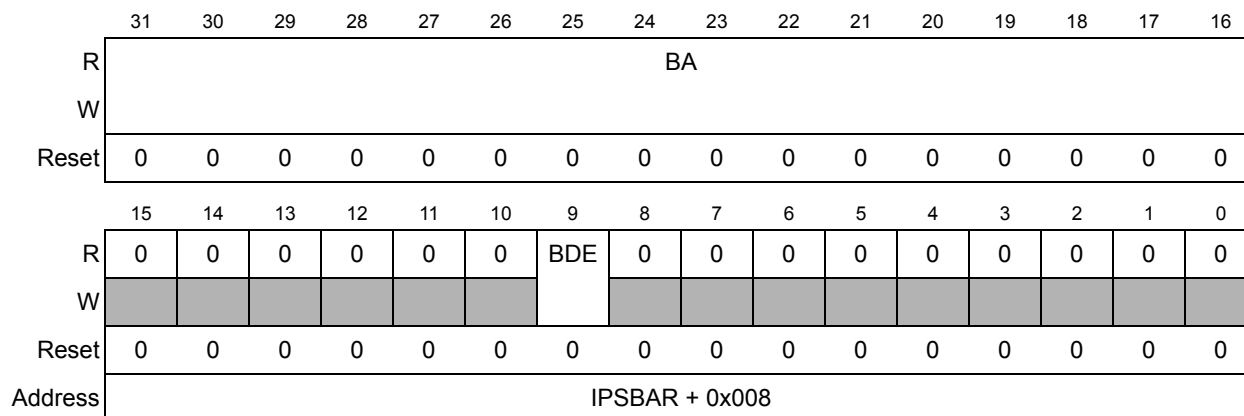
Bits	Name	Description
31–30	BA	Base address. Defines the base address of the 1-Gbyte internal peripheral space. This is the starting address for the IPS registers when the valid bit is set.
29–1	—	Reserved, should be cleared.
0	V	Valid. Enables/disables the IPS Base address region. V is set at reset. 0 IPS Base address is not valid. 1 IPS Base address is valid.

### 11.2.1.2 Memory Base Address Register (RAMBAR)

The MCF5275 supports dual-ported local SRAM memory. This processor-local memory can be accessed directly by the core and/or other system bus masters. Since this memory provides single-cycle accesses at processor speed, it is ideal for applications where double-buffer schemes can be used to maximize system-level performance. For example, a DMA channel in a typical double-buffer (also known as a ping-pong scheme) application may load data into one portion of the dual-ported SRAM while the processor is manipulating data in another portion of the SRAM. Once the processor completes the data calculations, it begins processing the just-loaded buffer while the DMA moves out the just-calculated data from the other buffer, and reloads the next data block into the just-freed memory region. The process repeats with the processor and the DMA “ping-ponging” between alternate regions of the dual-ported SRAM.

The MCF5275 design implements the dual-ported SRAM in the memory space defined by the RAMBAR register. There are two physical copies of the RAMBAR register: one located in the processor core and accessible only via the privileged MOVEC instruction at CPU space address 0xC05, and another located in the SCM at IPSBAR + 0x008. ColdFire core accesses to this memory are controlled by the processor-local copy of the RAMBAR, while (e.g., DMA, FEC, etc.) module accesses are enabled by the SCM's RAMBAR.

The physical base address programmed in both copies of the RAMBAR is typically the same value; however, they can be programmed to different values. By definition, the base address must be a 0-modulo-size value.



**Figure 11-2. Memory Base Address Register (RAMBAR)**

**Table 11-3. RAMBAR Field Description**

Bits	Name	Description
31–16	BA	Base address. Defines the memory module's base address on a 64-Kbyte boundary corresponding to the physical array location within the 4 Gbyte address space supported by ColdFire.
15–10	—	Reserved, should be cleared.
9	BDE	Back door enable. Qualifies the DMA module accesses to the SRAM memory. 0 Disables DMA module accesses to the SRAM module. 1 Enables DMA module accesses to the SRAM module. NOTE: The SPV bit in the CPU's RAMBAR must also be set to allow dual port access to the SRAM. For more information, see <a href="#">Section 6.2.1, "SRAM Base Address Register (RAMBAR)."</a>
8–0	—	Reserved, should be cleared.

The SRAM module is configured through the RAMBAR shown in [Figure 11-2](#).

- RAMBAR specifies the base address of the SRAM.
- All undefined bits are reserved. These bits are ignored during writes to the RAMBAR and return zeros when read.
- The back door enable bit, RAMBAR[BDE], is cleared at reset, disabling the DMA module access to the SRAM.

**NOTE**

The SCM RAMBAR default value of 0x0000\_0000 is invalid. The RAMBAR located in the processor's CPU space must be initialized with the valid bit, RAMBAR[V], set before the CPU (or modules) can access the on-chip SRAM (see 6.2.1, “SRAM Base Address Register (RAMBAR)” for more information. The SCM RAMBAR is implemented as 32 bits, all bits may be written and read. Bit fields [15:10] and [8:0] are not used in the access decode.

For details on the processor's view of the local SRAM memories, see Section 6.2.1, “SRAM Base Address Register (RAMBAR).”

### 11.2.1.3 Core Reset Status Register (CRSR)

The CRSR contains a bit for two of the reset sources to the CPU. A bit set to 1 indicates the last type of reset that occurred. The CRSR is updated by the control logic when the reset is complete. Only one bit is set at any one time in the CRSR. The register reflects the cause of the most recent reset. To clear a bit, a logic 1 must be written to the bit location; writing a zero has no effect.

**NOTE**

The reset status register (RSR) in the reset controller module (see Chapter 10, “Reset Controller Module”) provides indication of all reset sources except the core watchdog timer.

	7	6	5	4	3	2	1	0
R	EXT	0	0	0	0	0	0	0
W								
Reset	See Note							
Address	IPSBAR + 0x010							

**Note:** The reset value of EXT depends on the last reset source. All other bits are initialized to zero.

**Figure 11-3. Core Reset Status Register (CRSR)**

**Table 11-4. CRSR Field Descriptions**

Bits	Name	Description
7	EXT	External reset. 1 An external device driving $\overline{\text{RESET}}$ caused the last reset. Assertion of reset by an external device causes the processor core to initiate reset exception processing. All registers are forced to their initial state.
6–0	—	Reserved, should be cleared.



### 11.2.1.4 Core Watchdog Control Register (CWCR)

The core watchdog timer prevents system lockup if the software becomes trapped in a loop with no controlled exit. The core watchdog timer can be enabled or disabled through CWCR[CWE]. By default it is disabled. If enabled, the watchdog timer requires the periodic execution of a core watchdog servicing sequence. If this periodic servicing action does not occur, the timer times out, resulting in a watchdog timer interrupt. If the timer times out and the core watchdog transfer acknowledge enable bit (CWCR[CWTA]) is set, a watchdog timer interrupt is asserted. If a core watchdog timer interrupt acknowledge cycle has not occurred after another timeout, CWT TA is asserted in an attempt to allow the interrupt acknowledge cycle to proceed by terminating the bus cycle. The setting of CWCR[CWTAVAL] indicates that the watchdog timer TA was asserted.

To prevent the core watchdog timer from interrupting, the CWSR must be serviced by performing the following sequence:

1. Write 0x55 to CWSR.
2. Write 0xAA to the CWSR.

Both writes must occur in order before the time-out, but any number of instructions can be executed between the two writes. This order allows interrupts and exceptions to occur, if necessary, between the two writes. Caution should be exercised when changing CWCR values after the software watchdog timer has been enabled with the setting of CWCR[CWE], because it is difficult to determine the state of the core watchdog timer while it is running. The countdown value is constantly compared with the time-out period specified by CWCR[CWT]. The following steps must be taken to change CWT:

1. Disable the core watchdog timer by clearing CWCR[CWE].
2. Reset the counter by writing 0x55 and then 0xAA to CWSR.
3. Update CWCR[CWT].
4. Re-enable the core watchdog timer by setting CWCR[CWE]. This step can be performed in step 3.

The CWCR controls the software watchdog timer, time-out periods, and software watchdog timer transfer acknowledge. The register can be read at any time, but can be written only if the CWT is not pending. At system reset, the software watchdog timer is disabled.

	7	6	5	4	3	2	1	0
R	CWE	CWRI	CWT			CWTA	CWTAV AL	CWTIC
W								
Reset	0	0	0	0	0	0	0	0
Address	IPSBAR + 0x011							

**Figure 11-4. Core Watchdog Control Register (CWCR)**

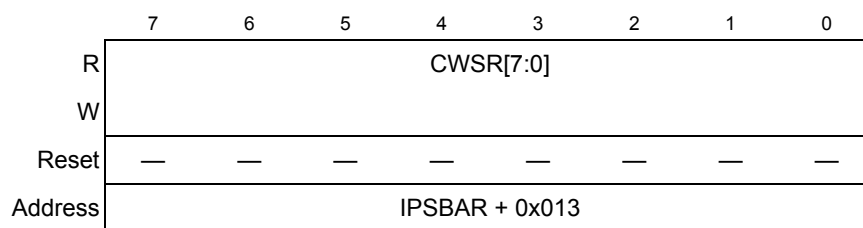
**Table 11-5. CWCR Field Description**

Bits	Name	Description																		
7	CWE	Core watchdog enable. 0 SWT disabled. 1 SWT enabled.																		
6	CWRI	Core watchdog reset/interrupt select. 0 If a time-out occurs, the CWT generates an interrupt to the processor core. The interrupt level for the CWT is programmed in the interrupt control register 7 (ICR7) of INTC0. 1 Reserved. Please note that unlike legacy devices, this bit is not available since the core watchdog is unable to reset the device.																		
5–3	CWT	Core watchdog timing delay. These bits select the timeout period for the CWT. At system reset, the CWT field is cleared signaling the minimum time-out period but the watchdog is disabled (CWCR[CWE] = 0). <table><tr><th>CWT [2:0]</th><th>CWT Time-Out Period</th></tr><tr><td>000</td><td>2<sup>9</sup> Bus clock frequency</td></tr><tr><td>001</td><td>2<sup>11</sup> Bus clock frequency</td></tr><tr><td>010</td><td>2<sup>13</sup> Bus clock frequency</td></tr><tr><td>011</td><td>2<sup>15</sup> Bus clock frequency</td></tr><tr><td>100</td><td>2<sup>19</sup> Bus clock frequency</td></tr><tr><td>101</td><td>2<sup>23</sup> Bus clock frequency</td></tr><tr><td>110</td><td>2<sup>27</sup> Bus clock frequency</td></tr><tr><td>111</td><td>2<sup>31</sup> Bus clock frequency</td></tr></table>	CWT [2:0]	CWT Time-Out Period	000	2 <sup>9</sup> Bus clock frequency	001	2 <sup>11</sup> Bus clock frequency	010	2 <sup>13</sup> Bus clock frequency	011	2 <sup>15</sup> Bus clock frequency	100	2 <sup>19</sup> Bus clock frequency	101	2 <sup>23</sup> Bus clock frequency	110	2 <sup>27</sup> Bus clock frequency	111	2 <sup>31</sup> Bus clock frequency
CWT [2:0]	CWT Time-Out Period																			
000	2 <sup>9</sup> Bus clock frequency																			
001	2 <sup>11</sup> Bus clock frequency																			
010	2 <sup>13</sup> Bus clock frequency																			
011	2 <sup>15</sup> Bus clock frequency																			
100	2 <sup>19</sup> Bus clock frequency																			
101	2 <sup>23</sup> Bus clock frequency																			
110	2 <sup>27</sup> Bus clock frequency																			
111	2 <sup>31</sup> Bus clock frequency																			
2	CWTA	Core watchdog transfer acknowledge enable. 0 CWTA Transfer acknowledge disabled. 1 CWTA Transfer Acknowledge enabled. After one CWT time-out period of the unacknowledged assertion of the CWT interrupt, the transfer acknowledge asserts, which allows CWT to terminate a bus cycle and allow the interrupt acknowledge to occur.																		
1	CWTAVAL	Core watchdog transfer acknowledge valid. 0 CWTA Transfer Acknowledge has not occurred. 1 CWTA Transfer Acknowledge has occurred. Write a 1 to clear this flag bit.																		
0	CWTIF	Core watchdog timer interrupt flag. 0 CWT interrupt has not occurred 1 CWT interrupt has occurred. Write a 1 to clear the interrupt request.																		

### 11.2.1.5 Core Watchdog Service Register (CWSR)

The software watchdog service sequence must be performed using the CWSR as a data register to prevent a CWT time-out. The service sequence requires two writes to this data register: first a write of 0x55 followed by a write of 0xAA. Both writes must be performed in this order prior to the CWT time-out, but any number of instructions or accesses to the CWSR can be executed between

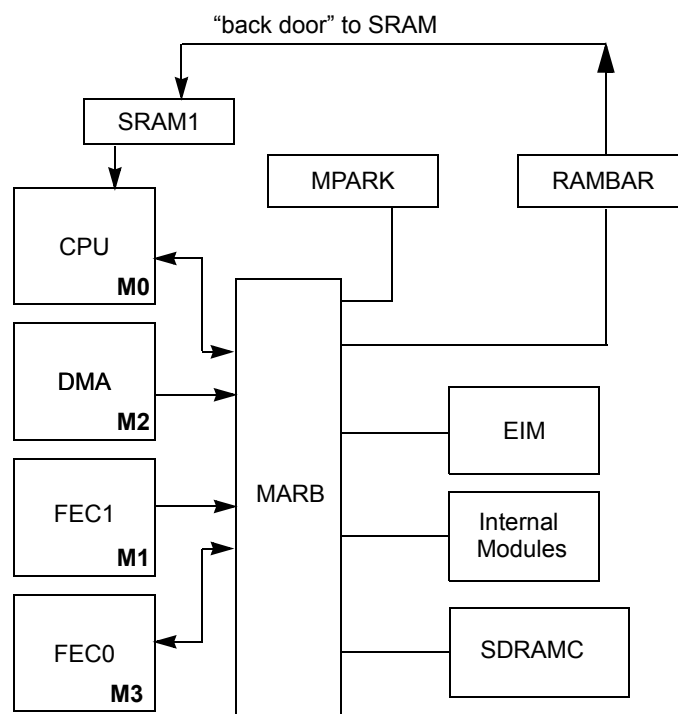
the two writes. If the CWT has already timed out, writing to this register has no effect in negating the CWT interrupt. Figure 11-5 illustrates the CWSR. At system reset, the contents of CWSR are uninitialized.



**Figure 11-5. Core Watchdog Service Register (CWSR)**

## 11.3 Internal Bus Arbitration

The internal bus arbitration is performed by the on-chip bus arbiter, which containing the arbitration logic that controls which of up to four MBus masters (M0–M3 in Figure 11-6) has access to the external buses. The function of the arbitration logic is described in this section.



**Figure 11-6. Arbiter Module Functions**

## 11.3.1 Overview

The basic functionality is that of a 4-port, pipelined internal bus arbitration module with the following attributes:

- The master pointed to by the current arbitration pointer may get on the bus with zero latency if the address phase is available. All other requesters face at least a one cycle arbitration pipeline delay in order to meet bus timing constraints on address phase hold.
- If a requester will get an immediate address phase (that is, it is pointed to by the current arbitration pointer and the bus address phase is available), it will be the current bus master and is ignored by arbitration. All remaining requesting ports are evaluated by the arbitration algorithm to determine the next-state arbitration pointer.
- There are two arbitration algorithms, fixed and round-robin. Fixed arbitration sets the next-state arbitration pointer to the highest priority requester. Round-robin arbitration sets the next-state arbitration pointer to the highest priority requester (calculated by adding a requester's fixed priority to the current bus master's fixed priority and then taking this sum modulo the number of possible bus masters).
- The default priority is FEC0/1 (M3) > DMA (M2) > internal master (M1) > CPU (M0), where M3 is the highest and M0 the lowest priority.
- There are two actions for an idle arbitration cycle, either leave the current arbitration pointer as is or set it to the lowest priority requester.
- The anti-lock-out logic for the fixed priority scheme forces the arbitration algorithm to round-robin if any requester has been held for longer than a specified cycle count.

## 11.3.2 Arbitration Algorithms

There are two modes of arbitration: fixed and round-robin. This section discusses the differences between them.

### 11.3.2.1 Round-Robin Mode

Round-robin arbitration is the default mode after reset. This scheme cycles through the sequence of masters as specified by MPARK[M<sub>n</sub>\_PTY] bits. Upon completion of a transfer, the master is given the lowest priority and the priority for all other masters is increased by one.

	M3 = 11 M2 = 01 M1 = 10 M0 = 00
next +1	M3 = 00 M2 = 10 M1 = 11 M0 = 01
next +2	M3 = 01 M2 = 11 M1 = 00 M0 = 10
next +3	M3 = 10 M2 = 00 M1 = 01 M0 = 11

If no masters are requesting, the arbitration unit must “park”, pointing at one of the masters. There are two possibilities, park the arbitration unit on the last active master, or park pointing to the

highest priority master. Setting MPARK[PRK\_LAST] causes the arbitration pointer to be parked on the highest priority master. In round-robin mode, programming the timeout enable and lockout bits MPARK[13,11:8] will have no effect on the arbitration.

### 11.3.2.2 Fixed Mode

In fixed arbitration the master with highest priority (as specified by the MPARK[Mn\_PRTY] bits) will win the bus. That master will relinquish the bus when all transfers to that master are complete.

If MPARK[TIMEOUT] is set, a counter will increment for each master for every cycle it is denied access. When a counter reaches the limit set by MPARK[LCKOUT\_TIME], the arbitration algorithm will be changed to round-robin arbitration mode until all locks are cleared. The arbitration will then return to fixed mode and the highest priority master will be granted the bus.

As in round-robin mode, if no masters are requesting, the arbitration pointer will park on the highest priority master if MPARK[PRK\_LAST] is set, or will park on the master which last requested the bus if cleared.

### 11.3.3 Bus Master Park Register (MPARK)

The MPARK controls the operation of the system bus arbitration module. The platform bus master connections are defined as:

- Master 3 (M3): Fast Ethernet Controller 1 (FEC0)
- Master 2 (M2): 4-channel DMA
- Master 1 (M1): Fast Ethernet Controller 2 (FEC1)
- Master 0 (M0): V2 ColdFire Core

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	M2_P	0	M3_PRTY	M2_PRTY	M0_PRTY	M1_PRTY				
W							_EN									
Reset	0	0	1	1	0	0	0	0	1	1	1	0	0	0	0	1
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	FIXED	TIME	PRK	LCKOUT_TIME				0	0	0	0	0	0	0	0
W			OUT	LAST												
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Address	IPSBAR + 0x01C															

**Figure 11-7. Default Bus Master Park Register (MPARK)**

**Table 11-6. MPARK Field Description**

Bits	Name	Description
31–26	—	Reserved, should be cleared.
25	M2_P_EN	DMA bandwidth control enable 0 disable the use of the DMA's bandwidth control to elevate the priority of its bus requests. 1 enable the use of the DMA's bandwidth control to elevate the priority of its bus requests.
24	—	Reserved, should be cleared.
23–22	M3_PRTY	Master priority level for master 3 (Fast Ethernet Controller 1) 00 fourth (lowest) priority 01 third priority 10 second priority 11 first (highest) priority
21–20	M2_PRTY	Master priority level for master 2 (DMA Controller) 00 fourth (lowest) priority 01 third priority 10 second priority 11 first (highest) priority
19–18	M0_PRTY	Master priority level for master 0 (ColdFire Core) 00 fourth (lowest) priority 01 third priority 10 second priority 11 first (highest) priority
17–16	M1_PRTY	Master priority level for master 1 (Fast Ethernet Controller 2) 00 fourth (lowest) priority 01 third priority 10 second priority 11 first (highest) priority
15	—	Reserved, should be cleared.
14	FIXED	Fixed or round robin arbitration 0 round robin arbitration 1 fixed arbitration
13	TIMEOUT	Timeout Enable 0 disable count for when a master is locked out by other masters. 1 enable count for when a master is locked out by other masters and allow access when LCKOUT_TIME is reached.
12	PRKLAST	Park on the last active master or highest priority master if no masters are active 0 park on last active master 1 park on highest priority master
11–8	LCKOUT_TIME	Lock-out Time. Lock-out time for a master being denied the bus. The lock out time is defined as $2^{\text{LCKOUT\_TIME}[3:0]}$ .
7–0	—	Reserved, should be cleared.

The initial state of the master priorities is  $M3 > M2 > M1 > M0$ . System software should guarantee that the programmed  $Mn\_PRTY$  fields are unique, otherwise the hardware defaults to the initial-state priorities.

**NOTE**

The M1\_PRTY field should not be set for a priority higher than third (default).

## 11.4 System Access Control Unit (SACU)

This section details the functionality of the System Access Control Unit (SACU) which provides the mechanism needed to implement secure bus transactions to the address space mapped to the internal modules.

### 11.4.1 Overview

The SACU supports the traditional model of two privilege levels: supervisor and user. Typically, memory references with the supervisor attribute have total accessibility to all the resources in the system, while user mode references cannot access system control and configuration registers. In many systems, the operating system executes in supervisor mode, while application software executes in user mode.

The SACU further partitions the access control functions into two parts: one control register defines the privilege level associated with each bus master, and another set of control registers define the access levels associated with the peripheral modules and the memory space.

The SACU's programming model is physically implemented as part of the System Control Module (SCM) with the actual access control logic included as part of the arbitration controller. Each bus transaction targeted for the IPS space is first checked to see if its privilege rights allow access to the given memory space. If the privilege rights are correct, the access proceeds on the bus. If the privilege rights are insufficient for the targeted memory space, the transfer is immediately aborted and terminated with an exception, and the targeted module not accessed.

### 11.4.2 Features

Each bus transfer can be classified by its privilege level and the reference type. The complete set of access types includes:

- Supervisor instruction fetch
- Supervisor operand read
- Supervisor operand write
- User instruction fetch
- User operand read
- User operand write

Instruction fetch accesses are associated with the execute attribute.

It should be noted that while the bus does not implement the concept of reference type (code versus data) and only supports the user/supervisor privilege level, the reference type attribute is supported by the system bus. Accordingly, the access checking associated with both privilege level and reference type is performed in the IPS controller using the attributes associated with the reference from the system bus.

The SACU partitions the access control mechanisms into three distinct functions:

- Master privilege register (MPR)
  - Allows each bus master to be assigned a privilege level:
    - Disable the master’s user/supervisor attribute and force to user mode access
    - Enable the master’s user/supervisor attribute
  - The reset state provides supervisor privilege to the processor core (bus master 0).
  - Input signals allow the non-core bus masters to have their user/supervisor attribute enabled at reset. This is intended to support the concept of a trusted bus master, and also controls the ability of a bus master to modify the register state of any of the SACU control registers; that is., only trusted masters can modify the control registers.
- Peripheral access control registers (PACRs)
  - Nine 8-bit registers control access to 17 of the on-chip peripheral modules.
  - Provides read/write access rights, supervisor/user privilege levels
  - Reset state provides supervisor-only read/write access to these modules
- Grouped peripheral access control registers (GPACR0)
  - One single register (GPACR) controls access to 14 of the on-chip peripheral modules
  - Provide read/write/execute access rights, supervisor/user privilege levels
  - Reset state provides supervisor-only read/write access to each of these peripheral spaces

### 11.4.3 Memory Map/Register Definition

The memory map for the SACU program-visible registers within the System Control Module (SCM) is shown in [Figure 11-7](#). The MPR, PACR, and GPACR are 8 bits in width.

**Table 11-7. SACU Register Memory Map**

IPSBA R Offset	[31:28]	[27:24]	[23:20]	[19:16]	[15:12]	[11:8]	[7:4]	[3:0]
0x020	MPR		—	—	—	—	—	—
0x024	PACR0		PACR1		PACR2		PACR3	
0x028	PACR4		—		PACR5		PACR6	
0x02c	PACR7		—		PACR8		—	



**Table 11-7. SACU Register Memory Map (Continued)**

IPSBA R Offset	[31:28]	[27:24]	[23:20]	[19:16]	[15:12]	[11:8]	[7:4]	[3:0]
0x030	GPACR		—		—		—	
0x034	—		—		—		—	
0x038	—		—		—		—	
0x03C	—		—		—		—	

### 11.4.3.1 Master Privilege Register (MPR)

The MPR specifies the access privilege level associated with each bus master in the platform. The register provides one bit per bus master, where bit 3 corresponds to master 3 (Fast Ethernet Controller 1), bit 2 to master 2 (DMA Controller), bit 1 to master 1 (Fast Ethernet Controller 2), and bit 0 to master 0 (ColdFire core).

	7	6	5	4	3	2	1	0
R	0	0	0	0	MPR3	MPR2	MPR1	MPR0
W								
Reset	0	0	0	0	0	0	1	1
Address	IPSBAR + 0x020							

**Figure 11-8. Master Privilege Register (MPR)****Table 11-8. MPR Field Descriptions**

Bits	Name	Description
7–4	—	Reserved. Should be cleared.
3–0	MPR <sub>n</sub>	Each 1-bit field defines the access privilege level of the given bus master <i>n</i> . 0 All bus master accesses are in user mode. 1 All bus master accesses use the sourced user/supervisor attribute.

Only trusted bus masters can modify the access control registers. If a non-trusted bus master attempts to write any of the SACU control registers, the access is aborted with an error termination and the registers remain unaffected.

The processor core is connected to bus master 0 and is always treated as a trusted bus master. Accordingly, MPR0 is forced to 1 at reset.

### 11.4.3.2 Peripheral Access Control Registers (PACR0–PACR8)

Access to several on-chip peripherals is controlled by shared peripheral access control registers. A single PACR defines the access level for each of the two modules. These modules only support

operand reads and writes. Each PACR follows the format illustrated in [Figure 11-10](#). For a list of PACRs and the modules that they control, refer to [Table 11-11](#).

	7	6	5	4	3	2	1	0
R	LOCK1	ACCESS_CTRL1			LOCK0	ACCESS_CTRL0		
W								
Reset	0	0	0	0	0	0	0	0
Address	IPSBAR + 0x24 + Offset							

**Figure 11-9. Peripheral Access Control Register (PACR<sub>n</sub>)**

**Table 11-9. PACR<sub>n</sub> Field Descriptions**

Bits	Name	Description
7	LOCK1	This bit, when set, prevents subsequent writes to ACCESSCTRL1. Any attempted write to the PACR generates an error termination and the contents of the register are not affected. Only a system reset clears this flag.
6–4	ACCESS_CTRL1	This 3-bit field defines the access control for the given platform peripheral. The encodings for this field are shown in <a href="#">Table 11-10</a> .
3	LOCK0	This bit, when set, prevents subsequent writes to ACCESSCTRL0. Any attempted write to the PACR generates an error termination and the contents of the register are not affected. Only a system reset clears this flag.
2–0	ACCESS_CTRL0	This 3-bit field defines the access control for the given platform peripheral. The encodings for this field are shown in <a href="#">Table 11-10</a> .

**Table 11-10. PACR ACCESS\_CTRL Bit Encodings**

Bits	Supervisor Mode		User Mode	
	Read	Write	Read	Write
000	x	x	—	—
001	x	—	—	—
010	x	—	x	—
011	x	—	—	—
100	x	x	x	x
101	x	x	x	—
110	x	x	x	x
111	—	—	—	—

**Table 11-11. Peripheral Access Control Registers (PACR<sub>n</sub>)**

IPSBAR Offset	Name	Modules Controlled	
		ACCESS_CTRL1	ACCESS_CTRL0
0x024	PACR0	SCM	SDRAMC
0x025	PACR1	EIM	DMA
0x026	PACR2	UART0	UART1
0x027	PACR3	UART2	—
0x028	PACR4	I <sup>2</sup> C	QSPI
0x029	—	—	—
0x02A	PACR5	DTIM0	DTIM1
0x02B	PACR6	DTIM2	DTIM3
0x02C	PACR7	INTC0	INTC1
0x02D	—	—	—
0x02E	PACR8	FEC0	FEC1

At reset, these on-chip modules are configured to have only supervisor read/write access capabilities. If an instruction fetch access to any of these peripheral modules is attempted, the IPS bus cycle is immediately terminated with an error.

#### 11.4.3.3 Grouped Peripheral Access Control Register (GPACR)

The on-chip peripheral space starting at IPSBAR is subdivided into sixteen 64-Mbyte regions. The first region has a unique access control register associated with it. The other fifteen regions are in reserved space; the access control registers for these regions are not implemented. The access control register is 8 bits in width so that read, write, and execute attributes may be assigned to the given IPS region.

#### NOTE

The access control for modules with memory space protected by PACR0–PACR8 are determined by the PACR0–PACR8 settings. The access control is not affected by GPACR, even though the modules are mapped in its 64-Mbyte address space.

	7	6	5	4	3	2	1	0
R	LOCK	0	0	0	ACCESS_CTRL			
W								
Reset	0	0	0	0	0	0	0	0
Address	IPSBAR + 0x030							

**Figure 11-10. Grouped Peripheral Access Control Register (GPACR)****Table 11-12. (GPACR) Field Descriptions**

Bits	Name	Description
7	LOCK	This bit, once set, prevents subsequent writes to the GPACR. Any attempted write to the GPACR generates an error termination and the contents of the register are not affected. Only a system reset clears this flag.
6–4	—	Reserved, should be cleared.
3–0	ACCESS_CTRL	This 4-bit field defines the access control for the given memory region. The encodings for this field are shown in <a href="#">Table 11-13</a> .

At reset, these on-chip modules are configured to have only supervisor read/write access capabilities. Bit encodings for the ACCESS\_CTRL field in the GPACR are shown in [Table 11-13](#). [Table 11-14](#) shows the memory space protected by the GPACR and the modules mapped to this space.

**Table 11-13. GPACR ACCESS\_CTRL Bit Encodings**

Bits	Supervisor Mode			User Mode		
	Read	Write	Execute	Read	Write	Execute
0000	x	x	—	—	—	—
0001	x	—	—	—	—	—
0010	x	—	—	x	—	—
0011	x	—	—	—	—	—
0100	x	x	—	x	x	—
0101	x	x	—	x	—	—
0110	x	x	—	x	x	—
0111	—	—	—	—	—	—
1000	x	x	x	—	—	—
1001	x	—	x	—	—	—
1010	x	—	x	x	—	x
1011	—	—	x	—	—	—
1100	x	x	x	x	x	x

**Table 11-13. GPACR ACCESS\_CTRL Bit Encodings (Continued)**

Bits	Supervisor Mode			User Mode		
	Read	Write	Execute	Read	Write	Execute
1101	x	x	x	x	—	x
1110	x	x	—	x	—	—
1111	x	x	x	—	—	x

**Table 11-14. GPACR Address Space**

Register	Space Protected (IPSBAR Offset)	Modules Protected
GPACR	0x0000_0000–0x03FF_FFFF	All



---

# Chapter 12

## General Purpose I/O Module

### 12.1 Introduction

Many of the pins associated with the MCF5275 external interface may be used for several different functions. Their primary functions are to provide external interfaces to access off-chip resources. When not used for their primary function, many of the pins may be used as general-purpose digital I/O pins. In some cases, the pin function is set by the operating mode, and the alternate pin functions are not supported.

The general purpose I/O pins on the MCF5275 are grouped into 8-bit ports. Some ports do not use all 8 bits. Each GPIO port has registers that configure, monitor, and control the port pins. [Figure 12-1](#) is a block diagram of the MCF5275 ports.

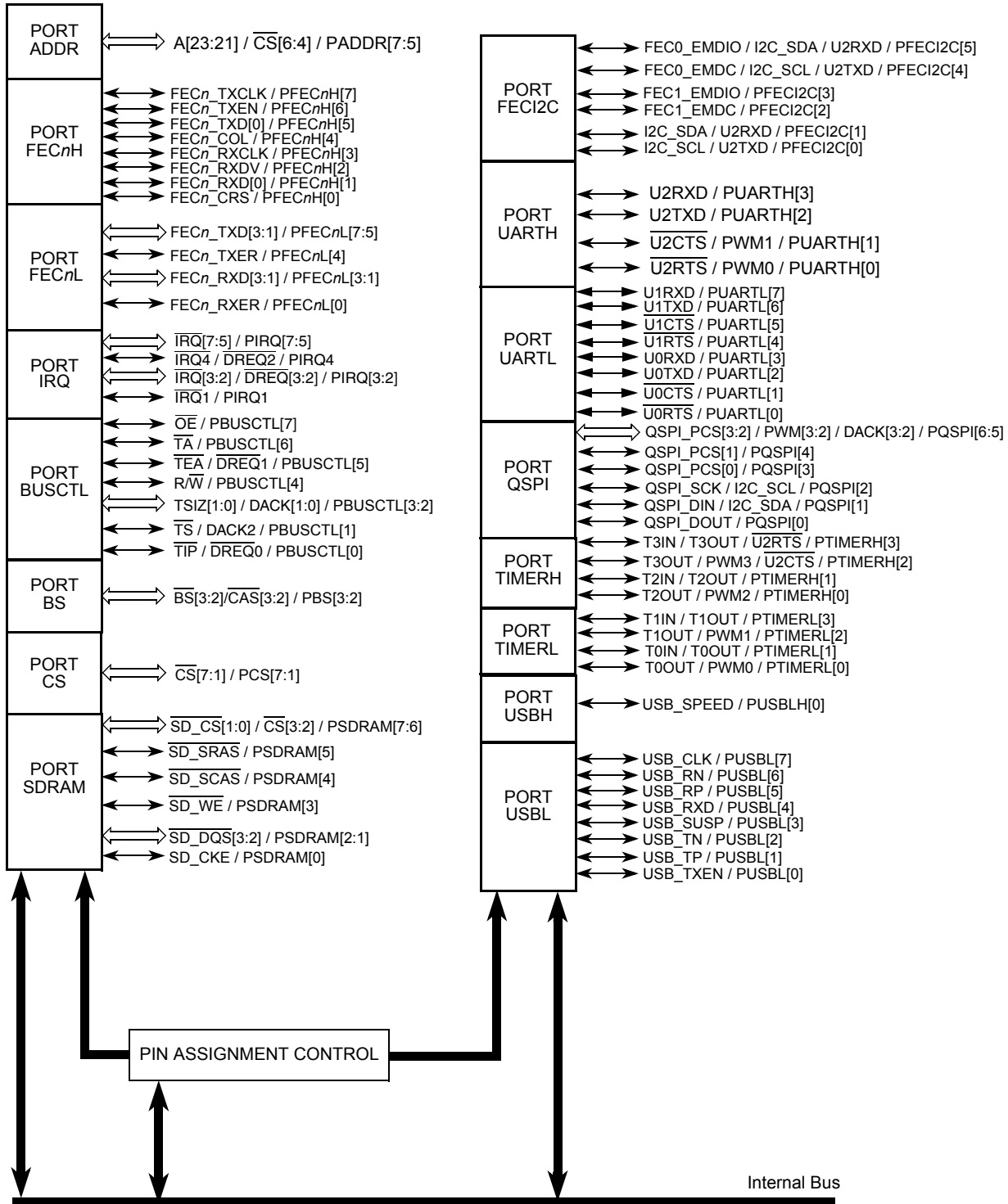


Figure 12-1. MCF5275 Ports Module Block Diagram



## 12.1.1 Overview

The MCF5275 ports module controls the configuration for various external pins, including those used for:

- External bus accesses
- External device selection
- Ethernet data and control
- I<sup>2</sup>C serial control
- QSPI
- 32-bit platform timers
- USB

## 12.1.2 Features

The MCF5275 ports includes these distinctive features:

- Control of primary function use
  - On all supported GPIO ports
- General purpose I/O support for all ports
  - Registers for storing output pin data
  - Registers for controlling pin data direction
  - Registers for reading current pin state
  - Registers for setting and clearing output pin data registers

## 12.2 External Signal Description

The MCF5275 ports control the functionality of several external pins. These pins are listed in [Table 12-2](#) under the GPIO column.

After reset ports ADDR, DATAH, DATAL, BUSCTL, BS and CS are configured for external memory. They are available for the user as GPIO if the corresponding registers are set appropriately. All other ports default to GPIO after reset.

### NOTE

In this table and throughout this document a single signal within a group is designated without square brackets (i.e., A24), while designations for multiple signals within a group use brackets (i.e., A[23:21]) and is meant to include all signals within the two bracketed numbers when these numbers are separated by a colon.

**NOTE**

The primary functionality of a pin is not necessarily its default functionality. Pins that are muxed with GPIO will default to their GPIO functionality.

**Table 12-1. MCF5274 and MCF5275 Signal Information and Muxing**

Signal Name	GPIO	Alternate1	Alternate2	Dir. <sup>1</sup>	MCF5274 MCF5275 256 MAPBGA	MCF5274L MCF5275L 196 MAPBGA
<b>Reset</b>						
$\overline{\text{RESET}}$	—	—	—	I	N15	K12
$\overline{\text{RSTOUT}}$	—	—	—	O	N14	L12
<b>Clock</b>						
EXTAL	—	—	—	I	L16	M14
XTAL	—	—	—	O	M16	N14
CLKOUT	—	—	—	O	T12	P9
<b>Mode Selection</b>						
CLKMOD[1:0]	—	—	—	I	N13, P13	M11, N11
$\overline{\text{RCON}}$	—	—	—	I	P8	M6
<b>External Memory Interface and Ports</b>						
A[23:21]	PADDR[7:5]	$\overline{\text{CS}}[6:4]$	—	O	A11, B11, C11	A8, B8, C8
A[20:0]	—	—	—	O	A12, B12, C12, A13, B13, C13, A14, B14, C14, B15, C15, B16, C16, D14, D15, E14:16, F14:16	B9, D9, C9, C10, B10, A11, C11, B11, A12, D11, C12, B13, C13, D12, E11, D13, E12, F11, D14, E13, F13
D[31:16]	—	—	—	O	M1, N1, N2, N3, P1, P2, R1, R2, P3, R3, T3, N4, P4, R4, T4, N5	J3, L1, K2, K3, M1, L2, L3, L4, K4, J4, M2, N1, N2, M3, M4, N3
$\overline{\text{BS}}[3:2]$	$\overline{\text{PBS}}[3:2]$	$\overline{\text{CAS}}[3:2]$	—	O	M3, R5	K1, L5
$\overline{\text{OE}}$	PBUSCTL[7]	—	—	O	K1	H4
$\overline{\text{TA}}$	PBUSCTL[6]	—	—	I	L13	K14
$\overline{\text{TEA}}$	PBUSCTL[5]	$\overline{\text{DREQ1}}$	—	I	T8	—
$\text{R}/\overline{\text{W}}$	PBUSCTL[4]	—	—	O	P7	L6
TSIZ1	PBUSCTL[3]	DACK1	—	O	D16	B14
TSIZ0	PBUSCTL[2]	DACK0	—	O	G16	E14

**Table 12-1. MCF5274 and MCF5275 Signal Information and Muxing (Continued)**

Signal Name	GPIO	Alternate1	Alternate2	Dir. <sup>1</sup>	MCF5274 MCF5275 256 MAPBGA	MCF5274L MCF5275L 196 MAPBGA
$\overline{TS}$	PBUSCTL[1]	DACK2	—	O	L4	H2
$\overline{TIP}$	PBUSCTL[0]	$\overline{DREQ0}$	—	O	P6	—
<b>Chip Selects</b>						
$\overline{CS}[7:1]$	PCS[7:1]	—	—	O	D10:13, E13, F13, N7	D8, A9, A10, D10, B12, C14, P4
$\overline{CS0}$	—	—	—	O	R6	N5
<b>DDR SDRAM Controller</b>						
DDR_CLKOUT	—	—	—	O	T7	P6
$\overline{DDR\_CLKOUT}$	—	—	—	O	T6	P5
$\overline{SD\_CS}[1:0]$	PSDRAM[7:6]	$\overline{CS}[3:2]$	—	O	M2, T5	H3, M5
$\overline{SD\_SRAS}$	PSDRAM[5]	—	—	O	L2	H1
$\overline{SD\_SCAS}$	PSDRAM[4]	—	—	O	L1	G3
$\overline{SD\_WE}$	PSDRAM[3]	—	—	O	K2	G4
SD_A10	—	—	—	O	N6	N4
$\overline{SD\_DQS}[3:2]$	PSDRAM[2:1]	—	—	I/O	M4, P5	J2, P3
SD_CKE	PSDRAM[0]	—	—	O	L3	J1
SD_VREF	—	—	—	I	A15, T2	A13, P2
<b>External Interrupts Port</b>						
$\overline{IRQ}[7:5]$	PIRQ[7:5]	—	—	I	G13, H16, H15	F14, G13, G14
$\overline{IRQ}[4]$	PIRQ[4]	$\overline{DREQ2}$	—	I	H14	H11
$\overline{IRQ}[3:2]$	PIRQ[3:2]	$\overline{DREQ}[3:2]$	—	I	J14, J13	H14, H12
$\overline{IRQ1}$	PIRQ[1]	—	—	I	K13	J13
<b>FEC0</b>						
FEC0_MDIO	PFECI2C[5]	I2C_SDA	U2RXD	I/O	A7	A3
FEC0_MDC	PFECI2C[4]	I2C_SCL	U2TXD	O	B7	C5
FEC0_TXCLK	PFEC0H[7]	—	—	I	C3	C1
FEC0_TXEN	PFEC0H[6]	—	—	O	D4	C3
FEC0_TXD[0]	PFEC0H[5]	—	—	O	G4	D2
FEC0_COL	PFEC0H[4]	—	—	I	A6	B4
FEC0_RXCLK	PFEC0H[3]	—	—	I	B6	B3

**Table 12-1. MCF5274 and MCF5275 Signal Information and Muxing (Continued)**

Signal Name	GPIO	Alternate1	Alternate2	Dir. <sup>1</sup>	MCF5274 MCF5275 256 MAPBGA	MCF5274L MCF5275L 196 MAPBGA
FEC0_RXDV	PFEC0H[2]	—	—	I	B5	C4
FEC0_RXD[0]	PFEC0H[1]	—	—	I	C6	D5
FEC0_CRS	PFEC0H[0]	—	—	I	C7	A2
FEC0_TXD[3:1]	PFEC0L[7:5]	—	—	O	E3, F3, F4	D1, E3, D3
FEC0_TXER	PFEC0L[4]	—	—	O	D3	C2
FEC0_RXD[3:1]	PFEC0L[3:1]	—	—	I	D5, C5, D6	D4, B1, B2
FEC0_RXER	PFEC0L[0]	—	—	I	C4	E4
<b>FEC1</b>						
FEC1_MDIO	PFEC1C[3]	—	—	I/O	G1	—
FEC1_MDC	PFEC1C[2]	—	—	O	G2	—
FEC1_TXCLK	PFEC1H[7]	—	—	I	C1	—
FEC1_TXEN	PFEC1H[6]	—	—	O	D2	—
FEC1_TXD[0]	PFEC1H[5]	—	—	O	F1	—
FEC1_COL	PFEC1H[4]	—	—	I	A5	—
FEC1_RXCLK	PFEC1H[3]	—	—	I	B4	—
FEC1_RXDV	PFEC1H[2]	—	—	I	A3	—
FEC1_RXD[0]	PFEC1H[1]	—	—	I	B3	—
FEC1_CRS	PFEC1H[0]	—	—	I	A4	—
FEC1_TXD[3:1]	PFEC1L[7:5]	—	—	O	E1, E2, F2	—
FEC1_TXER	PFEC1L[4]	—	—	O	D1	—
FEC1_RXD[3:1]	PFEC1L[3:1]	—	—	I	B1, B2, A2	—
FEC1_RXER	PFEC1L[0]	—	—	I	C2	—
<b>I<sup>2</sup>C</b>						
I2C_SDA	PFEC1C[1]	U2RXD	—	I/O	B10	B7
I2C_SCL	PFEC1C[0]	U2TXD	—	I/O	C10	A7
<b>DMA</b>						
DACK[3:0] and $\overline{\text{DREQ}}[3:0]$ do not have a dedicated bond pads. Please refer to the following pins for muxing: PCS3/PWM3 for DACK3, PCS2/PWM2 for DACK2, TSIZ1 for DACK1, TSIZ0 for DACK0, IRQ3 for $\overline{\text{DREQ}}3$ , IRQ2 and TA for $\overline{\text{DREQ}}2$ , TEA for $\overline{\text{DREQ}}1$ , and TIP for $\overline{\text{DREQ}}0$ .					—	—
<b>QSPI</b>						

**Table 12-1. MCF5274 and MCF5275 Signal Information and Muxing (Continued)**

Signal Name	GPIO	Alternate1	Alternate2	Dir. <sup>1</sup>	MCF5274 MCF5275 256 MAPBGA	MCF5274L MCF5275L 196 MAPBGA
QSPI_CS[3:2]	PQSPI[6:5]	PWM[3:2]	DACK[3:2]	O	R13, N12	P10, N9
QSPI_CS1	PQSPI[4]	—	—	O	T14	N10
QSPI_CS0	PQSPI[3]	—	—	O	P12	M9
QSPI_CLK	PQSPI[2]	I2C_SCL	—	O	T15	L11
QSPI_DIN	PQSPI[1]	I2C_SDA	—	I	T13	M10
QSPI_DOUT	PQSPI[0]	—	—	O	R12	L10
<b>UARTs</b>						
U2RXD	PUARTH[3]	—	—	I	T9	—
U2TXD	PUARTH[2]	—	—	O	R9	—
$\overline{\text{U2CTS}}$	PUARTH[1]	PWM1	—	I	P9	—
$\overline{\text{U2RTS}}$	PUARTH[0]	PWM0	—	O	R8	—
U1RXD	PUARTL[7]	—	—	I	A9	A6
U1TXD	PUARTL[6]	—	—	O	B9	D7
$\overline{\text{U1CTS}}$	PUARTL[5]	—	—	I	C9	C7
$\overline{\text{U1RTS}}$	PUARTL[4]	—	—	O	D9	B6
U0RXD	PUARTL[3]	—	—	I	A8	A4
U0TXD	PUARTL[2]	—	—	O	B8	A5
$\overline{\text{U0CTS}}$	PUARTL[1]	—	—	I	C8	C6
$\overline{\text{U0RTS}}$	PUARTL[0]	—	—	O	D7	B5
<b>USB</b>						
USB_SPEED	PUSBH[0]	—	—	I/O	G14	G11
USB_CLK	PUSBL[7]	—	—	I	G15	F12
USB_RN	PUSBL[6]	—	—	I	J16	H13
USB_RP	PUSBL[5]	—	—	I	J15	J11
USB_RXD	PUSBL[4]	—	—	I	L15	L14
USB_SUSP	PUSBL[3]	—	—	O	M13	N13
USB_TN	PUSBL[2]	—	—	O	K14	J14
USB_TP	PUSBL[1]	—	—	O	K15	J12
USB_TXEN	PUSBL[0]	—	—	O	L14	K13
<b>Timers (and PWMs)</b>						
DT3IN	PTIMERH[3]	DT3OUT	$\overline{\text{U2RTS}}$	I	J4	G2

**Table 12-1. MCF5274 and MCF5275 Signal Information and Muxing (Continued)**

Signal Name	GPIO	Alternate1	Alternate2	Dir. <sup>1</sup>	MCF5274 MCF5275 256 MAPBGA	MCF5274L MCF5275L 196 MAPBGA
DT3OUT	PTIMERH[2]	PWM3	U2CTS	O	K3	G1
DT2IN	PTIMERH[1]	DT2OUT	—	I	J2	F3
DT2OUT	PTIMERH[0]	PWM2	—	O	J3	F4
DT1IN	PTIMERL[3]	DT1OUT	—	I	H1	F1
DT1OUT	PTIMERL[2]	PWM1	—	O	H2	F2
DT0IN	PTIMERL[1]	DT0OUT	—	I	H3	E1
DT0OUT	PTIMERL[0]	PWM0	—	O	G3	E2
<b>BDM/JTAG<sup>2</sup></b>						
DSCLK	—	TRST	—	I	P14	P13
PSTCLK	—	TCLK	—	O	P16	P12
BKPT	—	TMS	—	I	R15	N12
DSI	—	TDI	—	I	R16	M12
DSO	—	TDO	—	O	P15	K11
JTAG_EN	—	—	—	I	R14	P11
DDATA[3:0]	—	—	—	O	P10, N10, P11, N11	M7, N7, P8, L9
PST[3:0]	—	—	—	O	T10, R10, T11, R11	P7, L8, M8, N8
<b>Test</b>						
TEST	—	—	—	I	N9	N6
PLL_TEST	—	—	—	I	M14	—
<b>Power Supplies</b>						
VDDPLL	—	—	—	I	M15	M13
VSSPLL	—	—	—	I	K16	L13
VSS	—	—	—	I	A1, A10, A16, E5, E12, F6, F11, G7:10, H7:10, J1, J7:10, K7:10, L6, L11, M5, N16, R7, T1, T16	F7, F8, G6:9, H6:9, J7, J8

**Table 12-1. MCF5274 and MCF5275 Signal Information and Muxing (Continued)**

Signal Name	GPIO	Alternate1	Alternate2	Dir. <sup>1</sup>	MCF5274 MCF5275 256 MAPBGA	MCF5274L MCF5275L 196 MAPBGA
OVDD	—	—	—	I	E6:8, F5, F7, F8, G5, G6, H5, H6, J11, J12, K11, K12, L9, L10, L12, M9:11	E5:7, F5, F6, H10, J9, J10, K8:10
VDD	—	—	—	I	D8, H13, K4, N8	D6, G5, G12, L7
SD_VDD	—	—	—	I	E9:11, F9, F10, F12, G11, G12, H11, H12, J5, J6, K5, K6, L5, L7, L8, M6, M7, M8	E8:10, F9, F10, G10, H5, J5, J6, K5:7

<sup>1</sup> Refers to pin's primary function. All pins which are configurable for GPIO have a pullup enabled in GPIO mode with the exception of PBUSCTL[7], PBUSCTL[4:0], PADDR, PBS, PSDRAM.

<sup>2</sup> If JTAG\_EN is asserted, these pins default to Alternate 1 (JTAG) functionality. The GPIO module is not responsible for assigning these pins.

Refer to the [Chapter 2, “Signal Descriptions,”](#) for more detailed descriptions of these pins and other pins not controlled by the ports module. The function of most of the pins (primary function, GPIO, etc.) is determined by the ports module pin assignment registers. Refer to [Section 2.3, “External Signal Descriptions”](#) for detailed descriptions of pin functions.

It should be noted from [Table 12-2](#) that there are several cases where a function is available on more than one pin. While it is possible to enable the function on more than one pin simultaneously, this type of programming should be avoided for input functions to prevent unexpected behavior. All multiple-pin functions are listed in [Table 12-2](#).

**Table 12-2. MCF5275 Multiple-Pin Functions**

Function	Direction	Associated Pins
Chip select 6 ( $\overline{\text{CS6}}$ )	O	$\overline{\text{CS6}}$ , A23
Chip select 5 ( $\overline{\text{CS5}}$ )	O	$\overline{\text{CS5}}$ , A22
Chip select 4 ( $\overline{\text{CS4}}$ )	O	$\overline{\text{CS4}}$ , A21
SDRAM Chip select 1 ( $\overline{\text{SD\_CS1}}$ )	O	$\overline{\text{CS3}}$ , $\overline{\text{SD\_CS1}}$
SDRAM Chip select 0 ( $\overline{\text{SD\_CS0}}$ )	O	$\overline{\text{CS2}}$ , $\overline{\text{SD\_CS0}}$
SDRAMC clock enable (SD_CKE)	O	SD_CKE, QSPI_CS1
I <sup>2</sup> C serial data (I2C_SDA)	I/O	I2C_SDA, QSPI_DIN, FEC0_MDIO
I <sup>2</sup> C serial clock (I2C_SCL)	I/O	I2C_SCL, QSPI_CLK, FEC0_MDC
DMA request 2 ( $\overline{\text{DREQ2}}$ )	I	$\overline{\text{IRQ2}}$ , $\overline{\text{DIRQ}}[4]$
DMA acknowledge 2 (DACK2)	O	$\overline{\text{TS}}$ , QSPI_PCS2

**Table 12-2. MCF5275 Multiple-Pin Functions (Continued)**

Function	Direction	Associated Pins
UART2 transmit data (U2TXD)	O	U2TXD, I2C_SCL, FEC0_MDC
UART2 receive data (U2RXD)	I	U2RXD, I2C_SDA, FEC0_MDIO
UART2 clear-to-send ( $\overline{U2CTS}$ )	I	DT3OUT, $\overline{U2CTS}$
UART2 request-to-send ( $\overline{U2RTS}$ )	O	DT3IN, $\overline{U2RTS}$
Timer output 3 (T3OUT)	O	T3OUT, T3IN
Timer output 2 (DT2OUT)	O	DT2OUT, DT2IN
Timer output 1 (DT1OUT)	O	DT1OUT, DT1IN
PWM output 3	O	T3OUT, QSPI_PCS3
PWM output 2	O	T2OUT, QSPI_PCS2
PWM output 1	O	T1OUT, $\overline{U2CTS}$
PWM output 0	O	T0OUT, $\overline{U2RTS}$

## 12.3 Memory Map/Register Definition

Table 12-3 summarizes all the registers in the MCF5275 ports address space.

**Table 12-3. MCF5275 Ports Module Memory Map**

IPSBAR Offset	[31:24]	[23:16]	[15:8]	[7:0]	Access <sup>1</sup>
Port Output Data Registers					
0x10_0000	Reserved <sup>2</sup>				S/U
0x10_0004	PODR_BUSCTL	PODR_ADDR	Reserved <sup>2</sup>		S/U
0x10_0008	PODR_CS	Reserved <sup>2</sup>	PODR_FEC0H	PODR_FEC0L	S/U
0x10_000C	PODR_FECI2C	PODR_QSPI	PODR_SDRAM	PODR_TIMERH	S/U
0x10_0010	PODR_TIMERL	PODR_UARTL	PODR_FEC1H	PODR_FEC1L	S/U
0x10_0014	PODR_BS	PODR_IRQ	PODR_USBH	PODR_USBL	S/U
0x10_0018	PODR_UTH	Reserved <sup>2</sup>			S/U
Port Data Direction Registers					
0x10_001C	Reserved <sup>2</sup>				S/U
0x10_0020	PDDR_BUSCTL	PDDR_ADDR	Reserved <sup>2</sup>		S/U
0x10_0024	PDDR_CS	Reserved <sup>2</sup>	PDDR_FEC0H	PDDR_FEC0L	S/U
0x10_0028	PDDR_FECI2C	PDDR_QSPI	PDDR_SDRAM	PDDR_TIMERH	S/U
0x10_002C	PDDR_TIMERL	PDDR_UARTL	PDDR_FEC1H	PDDR_FEC1L	S/U
0x10_0030	PDDR_BS	PDDR_IRQ	PDDR_USBH	PDDR_USBL	S/U



**Table 12-3. MCF5275 Ports Module Memory Map (Continued)**

IPSBAR Offset	[31:24]	[23:16]	[15:8]	[7:0]	Access <sup>1</sup>
0x10_0034	PDDR_UARTH	Reserved <sup>2</sup>			S/U
Port Pin Data/Set Data Registers					
0x10_0038	Reserved <sup>2</sup>				S/U
0x10_003C	PPDSDR_BUSCTL	PPDSDR_ADDR	Reserved <sup>2</sup>		S/U
0x10_0040	PPDSDR_CS	Reserved <sup>2</sup>	PPDSDR_FEC0H	PPDSDR_FEC0L	S/U
0x10_0044	PPDSDR_FECI2C	PPDSDR_QSPI	PPDSDR_SDRAM	PPDSDR_TIMERH	S/U
0x10_0048	PPDSDR_TIMERL	PPDSDR_UARTL	PPDSDR_FEC1H	PPDSDR_FEC1L	S/U
0x10_004C	PPDSDR_BS	PPDSDR_IRQ	PPDSDR_USBH	PPDSDR_USBL	S/U
0x10_0050	PPDSDR_UARTH	Reserved <sup>2</sup>			S/U
Port Clear Output Data Registers					
0x10_0054	Reserved <sup>3</sup>				S/U
0x10_0058	PCLRR_BUSCTL	PCLRR_ADDR	Reserved <sup>3</sup>		S/U
0x10_005C	PCLRR_CS	Reserved <sup>2</sup>	PCLRR_FEC0H	PCLRR_FEC0L	S/U
0x10_0060	PCLRR_FECI2C	PCLRR_QSPI	PCLRR_SDRAM	PCLRR_TIMERH	S/U
0x10_0064	PCLRR_TIMERL	PCLRR_UARTL	PCLRR_FEC1H	PCLRR_FEC1L	S/U
0x10_0068	PCLRR_BS	PCLRR_IRQ	PCLRR_USBH	PCLRR_USBL	S/U
0x10_006C	PCLRR_UARTH	Reserved <sup>2</sup>			S/U
Port Pin Assignment Registers					
0x10_0070	PAR_ADDR	PAR_CS	PAR_BUSCTL		S/U
0x10_0074	PAR_IRQ		PAR_USB		S/U
0x10_0078	PAR_FEC0HL	PAR_FEC1HL	PAR_TIMERH	PAR_TIMERL	S/U
0x10_007C	PAR_UART		PAR_QSPI		S/U
0x10_0080	PAR_SDRAM		PAR_FEC12C		S/U
0x10_0084	PAR_BS	Reserved <sup>2</sup>			S/U

<sup>1</sup> S/U = supervisor or user mode access.

<sup>2</sup> Writing to reserved address locations has no effect and reading returns 0s.

## 12.3.1 Register Descriptions

### 12.3.1.1 Port Output Data Registers (PODR\_x)

The PODR\_x registers store the data to be driven on the corresponding port pins when the pins are configured for general purpose output. The PODR\_x registers are each 8 bits wide, but not all ports

use all 8 bits. The register definitions for all ports are shown in [Figure 12-2](#) through [Figure 12-5](#). The PODR\_*x* registers are read/write. At reset, all implemented bits in the PODR\_*x* registers are set. Reserved bits always remain cleared.

Reading a PODR\_*x* register returns the current values in the register, not the port *x* pin values. To set bits in a PODR\_*x* register, write 1s to the PODR\_*x* bits, or write 1s to the corresponding bits in the PPDSDR\_*x* register. To clear bits in a PODR\_*x* register, write 0s to the PODR\_*x* bits, or write 0s to the corresponding bits in the PCLRR\_*x* register.

	7	6	5	4	3	2	1	0
R	PODR_ <i>x</i>							
W								
Reset	1	1	1	1	1	1	1	1
Address	IPSBAR + 0x10_0004 (PODR_BUSCTL); IPSBAR + 0x10_000A (PODR_FEC0H); IPSBAR + 0x10_000B (PODR_FEC0L); IPSBAR + 0x10_000E (PODR_SDRAM); IPSBAR + 0x10_00011 (PODR_UARTL); IPSBAR + 0x10_0012 (PODR_FEC1H); IPSBAR + 0x10_0013 (PODR_FEC1L); IPSBAR + 0x10_0017 (PODR_USBL)							

**Figure 12-2. Port *x* Output Data Registers (PODR\_*x*)**

	7	6	5	4	3	2	1	0
R	0	0	0	0	PODR_ <i>x</i>			
W								
Reset	0	0	0	0	1	1	1	1
Address	; IPSBAR + 0x10_000F (PODR_TIMERH); IPSBAR + 0x10_0010 (PODR_TIMERL); IPSBAR + 0x10_0018 (PODR_earth)							

**Figure 12-3. Port *x* Output Data Registers (PODR\_*x*)**

	7	6	5	4	3	2	1	0
R	PODR_ADDR			0	0	0	0	0
W								
Reset	1	1	1	0	0	0	0	0
Address	IPSBAR + 0x10_0005 (PODR_ADDR)							

**Figure 12-4. Port ADDR Output Data Register (PODR\_ADDR)**

	7	6	5	4	3	2	1	0
R	PODR_CS							0
W								
Reset	1	1	1	1	1	1	1	0
Address	IPSBAR + 0x10_0008 (PODR_CS)							

**Figure 12-5. Port CS Output Data Register (PODR\_CS)**

	7	6	5	4	3	2	1	0
R	0	0	PODR_FECI2C					
W								
Reset	0	0	1	1	1	1	1	1
Address	IPSBAR + 0x10_000C (PODR_FECI2C)							

**Figure 12-6. Port FECI2C Output Data Register (PODR\_FECI2C)**

	7	6	5	4	3	2	1	0
R	0	PODR_QSPI						
W								
Reset	0	1	1	1	1	1	1	1
Address	IPSBAR + 0x10_000D (PODR_QSPI)							

**Figure 12-7. Port QSPI Output Data Register (PODR\_QSPI)**

	7	6	5	4	3	2	1	0
R	0	0	0	0	PODR_BS		0	0
W								
Reset	0	0	0	0	1	1	0	0
Address	IPSBAR + 0x10_0014 (PODR_BS)							

**Figure 12-8. Port BS Output Data Register (PODR\_BS)**

	7	6	5	4	3	2	1	0
R	PODR_IRQ							0
W								
Reset	1	1	1	1	1	1	1	0
Address	IPSBAR + 0x10_0015 (PODR_IRQ)							

**Figure 12-9. Port IRQ Output Data Register (PODR\_IRQ)**

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	PODR_USBH
W								
Reset	0	0	0	0	0	0	0	1
Address	IPSBAR + 0x10_0016 (PODR_USBH)							

**Figure 12-10. Port USBH Output Data Register (PODR\_USBH)****Table 12-4. PODR\_x Field Descriptions**

Name	Description
—	Reserved, should be cleared.
PODR_x	Port x output data bits. 0 Drives 0 when the port x pin is general purpose output 1 Drives 1 when the port x pin is general purpose output

**Note:** See above figures for bit field positions.

### 12.3.1.2 Port Data Direction Registers (PDDR\_x)

The PDDRs control the direction of the port *x* pin drivers when the pins are configured for general purpose I/O. The PDDR\_x registers are each eight bits wide, but not all ports use all eight bits. The register definitions for all ports are shown in [Figure 12-11](#) through [Figure 12-19](#).

The PDDRs are read/write. At reset, all bits in the PDDRs are cleared. Setting any bit in a PDDR\_x register configures the corresponding port *x* pin as an output. Clearing any bit in a PDDR\_x register configures the corresponding pin as an input.

	7	6	5	4	3	2	1	0
R	PDDR_x							
W								
Reset	0	0	0	0	0	0	0	0
Address	IPSBAR + 0x10_0020 (PDDR_BUSCTL); IPSBAR + 0x10_0026 (PDDR_FEC0H); IPSBAR + 0x10_0027 (PDDR_FEC0L); IPSBAR + 0x10_002A (PDDR_SDRAM); IPSBAR + 0x10_002D (PDDR_UARTL); IPSBAR + 0x10_002E (PDDR_FEC1H); IPSBAR + 0x10_002F (PDDR_FEC1L); IPSBAR + 0x10_0033 (PDDR_USBL)							

**Figure 12-11. Port Data Direction Registers (PDDR\_x)**

	7	6	5	4	3	2	1	0
R	0	0	0	0	PDDR_x			
W								
Reset	0	0	0	0	0	0	0	0
Address	IPSBAR + 0x10_002B (PDDR_TIMERH); IPSBAR + 0x10_002C (PDDR_TIMERL); IPSBAR + 0x10_0034 (PDDR_UEARTH)							

**Figure 12-12. Port Data Direction Registers (PDDR\_x)**

	7	6	5	4	3	2	1	0
R	PDDR_ADDR			0	0	0	0	0
W								
Reset	0	0	0	0	0	0	0	0
Address	IPSBAR + 0x10_0021 (PDDR_ADDR)							

**Figure 12-13. Port ADDR Data Direction Register (PDDR\_ADDR)**

	7	6	5	4	3	2	1	0
R	PDDR_CS							0
W								
Reset	0	0	0	0	0	0	0	0
Address	IPSBAR + 0x10_0024 (PDDR_CS)							

**Figure 12-14. Port CS Data Direction Register (PDDR\_CS)**

	7	6	5	4	3	2	1	0
R	0	0	PDDR_FECI2C					
W								
Reset	0	0	0	0	0	0	0	0
Address	IPSBAR + 0x10_0028 (PDDR_FECI2C)							

**Figure 12-15. Port FECI2C Data Direction Register (PDDR\_FECI2C)**

	7	6	5	4	3	2	1	0
R	0	PDDR_QSPI						
W								
Reset	0	0	0	0	0	0	0	0
Address	IPSBAR + 0x10_0029 (PDDR_QSPI)							

**Figure 12-16. Port QSPI Data Direction Register (PDDR\_QSPI)**

	7	6	5	4	3	2	1	0
R	0	0	0	0	PDDR_BS		0	0
W								
Reset	0	0	0	0	0	0	0	0
Address	IPSBAR + 0x10_0030 (PDDR_BS)							

**Figure 12-17. Port BS Data Direction Register (PDDR\_BS)**

	7	6	5	4	3	2	1	0
R	PDDR_IRQ							0
W								
Reset	0	0	0	0	0	0	0	0
Address	IPSBAR + 0x10_0031 (PDDR_IRQ)							

**Figure 12-18. Port IRQ Data Direction Register (PDDR\_IRQ)**

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	PDDR_USBH
W								
Reset	0	0	0	0	0	0	0	0
Address	IPSBAR + 0x10_0032 (PDDR_USBH)							

**Figure 12-19. Port USBH Data Direction Register (PDDR\_USBH)****Table 12-5. PDDR\_x Field Descriptions**

Name	Description
—	Reserved, should be cleared.
PDDR_x	Port x output data direction bits. 1 Port x pin configured as output 0 Port x pin configured as input

**Note:** See above figures for bit field positions.

### 12.3.1.3 Port Pin Data/Set Data Registers (PPDSDR\_x)

The PPDSDR\_x registers reflect the current pin states and control the setting of output pins when the pin is configured for general purpose I/O. The PPDSDR\_x registers are each eight bits wide, but not all ports use all eight bits. The register definitions for all ports are shown in [Figure 12-20](#) through [Figure 12-28](#).

The PPDSDR\_x registers are read/write. At reset, the bits in the PPDSDR\_x registers are set to the current pin states. Reading a PPDSDR\_x register returns the current state of the port x pins. Setting a PPDSDR\_x register sets the corresponding bits in the PDDR\_x register. Writing 0s has no effect.

	7	6	5	4	3	2	1	0
R	PPDSDR_x							
W								
Reset	Current Pin State							
Address	IPSBAR + 0x10_003C (PPDSDR_BUSCTL); IPSBAR + 0x10_0042 (PPDSDR_FEC0H); IPSBAR + 0x10_0043 (PPDSDR_FEC0L); IPSBAR + 0x10_0046 (PPDSDR_SDRAM); IPSBAR + 0x10_0049 (PPDSDR_UARTL); IPSBAR + 0x10_004A (PPDSDR_FEC1H); IPSBAR + 0x10_004B (PPDSDR_FEC1L); IPSBAR + 0x10_004F (PPDSDR_USBL)							

Figure 12-20. Port Pin Data/Set Data Registers (PPDSDR\_x)

	7	6	5	4	3	2	1	0
R	0	0	0	0	PPDSDR_x			
W								
Reset	0	0	0	0	Current Pin State			
Address	IPSBAR + 0x10_0047 (PPDSDR_TIMERH); IPSBAR + 0x10_0048 (PPDSDR_TIMERL); IPSBAR + 0x10_0050 (PPDSDR_earth)							

Figure 12-21. Port Pin Data/Set Data Registers (PPDSDR\_x)

	7	6	5	4	3	2	1	0
R	PPDSDR_ADDR			0	0	0	0	0
W								
Reset	Current Pin State			0	0	0	0	0
Address	IPSBAR + 0x10_003D (PPDSDR_ADDR)							

Figure 12-22. Port ADDR Pin Data/Set Data Register (PPDSDR\_ADDR)

	7	6	5	4	3	2	1	0
R	PPDSDR_CS							0
W								
Reset	Current pin state							0
Address	IPSBAR + 0x10_0040 (PPDSDR_CS)							

Figure 12-23. Port CS Pin Data/Set Data Register (PPDSDR\_CS)

	7	6	5	4	3	2	1	0
R	0	0	PDDR_FECI2C					
W								
Reset	0	0	Current pin state					
Address	IPSBAR + 0x10_0044 (PPDSDR_FECI2C)							

**Figure 12-24. Port FECI2C Pin Data/Set Data Register (PPDSDR\_FECI2C)**

	7	6	5	4	3	2	1	0
R	0	PPDSDR_QSPI						
W								
Reset	0	Current Pin State						
Address	IPSBAR + 0x10_0045 (PPDSDR_QSPI)							

**Figure 12-25. Port QSPI Pin Data/Set Data Register (PPDSDR\_QSPI)**

	7	6	5	4	3	2	1	0
R	0	0	0	0	PPDSDR_BS		0	0
W								
Reset	0	0	0	0	Current Pin State		0	0
Address	IPSBAR + 0x10_004C (PPDSDR_BS)							

**Figure 12-26. Port BS Pin Data/Set Data Register (PPDSDR\_BS)**

	7	6	5	4	3	2	1	0
R	PPDSDR_IRQ							0
W								
Reset	Current pin state							0
Address	IPSBAR + 0x10_004D (PPDSDR_IRQ)							

**Figure 12-27. Port IRQ Pin Data/Set Data Register (PPDSDR\_IRQ)**

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	PPDSDR_USBH
W								
Reset	0	0	0	0	0	0	0	Current pin state
Address	IPSBAR + 0x10_004E (PPDSDR_USBH)							

**Figure 12-28. Port USBH Pin Data/Set Data Register (PPDSDR\_USBH)**



**Table 12-6. PPDSDR\_x Field Descriptions**

Name	Description
—	Reserved, should be cleared.
PPDSDR_x	Port x Pin Data/Set Data Bits. 0 Port x pin state is 0 (read) 1 Port x pin state is 1 (read); set corresponding PODR_x bit (write)

**Note:** See above figures for bit field positions.

### 12.3.1.4 Port Clear Output Data Registers (PCLRR\_x)

Clearing a PCLRR\_x register clears the corresponding bits in the PODR\_x register. Setting it has no effect. Reading the PCLRR\_x register returns 0s. Most PODR\_x registers have a full 8-bit implementation, as shown in [Figure 12-29](#). The remaining PODR\_x registers use fewer than eight bits. Their bit definitions are shown in [Figure 12-30](#) through [Figure 12-37](#).

The PCLRR\_x registers are read/write accessible.

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0
W	PCLRR_x							
Reset	0	0	0	0	0	0	0	0
Address	IPSBAR + 0x10_0058 (PCLRR_BUSCTL); IPSBAR + 0x10_005E (PCLRR_FEC0H); IPSBAR + 0x10_005F (PCLRR_FEC0L); IPSBAR + 0x10_0062 (PCLRR_SDRAM); IPSBAR + 0x10_0065 (PCLRR_UARTL); IPSBAR + 0x10_0066 (PCLRR_FEC1H); IPSBAR + 0x10_0067 (PCLRR_FEC1L); IPSBAR + 0x10_006B (PCLRR_USBL)							

**Figure 12-29. Port Clear Output Data Registers (PCLRR\_x)**

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0
W					PCLRR_x			
Reset	0	0	0	0	0	0	0	0
Address	IPSBAR + 0x10_0063 (PCLRR_TIMERH); IPSBAR + 0x10_0064 (PCLRR_TIMERL); IPSBAR + 0x10_006C (PCLRR_UEARTH)							

**Figure 12-30. Port Clear Output Data Registers (PCLRR\_x)**

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0
W	PCLRR_ADDR							
Reset	0	0	0	0	0	0	0	0
Address	IPSBAR + 0x10_0059 (PCLRR_ADDR)							

**Figure 12-31. Port ADDR Clear Output Data Register (PCLRR\_ADDR)()**

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0
W	PCLRR_CS							
Reset	0	0	0	0	0	0	0	0
Address	IPSBAR + 0x10_005C (PCLRR_CS)							

**Figure 12-32. Port CS Clear Output Data Register (PCLRR\_CS)**

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0
W		PCLRR_QSPI						
Reset	0	0	0	0	0	0	0	0
Address	IPSBAR + 0x10_0060 (PCLRR_QSPI)							

**Figure 12-33. Port QSPI Clear Output Data Register (PCLRR\_QSPI)**

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0
W			PCLRR_FECI2C					
Reset	0	0	0	0	0	0	0	0
Address	IPSBAR + 0x10_0060 (PCLRR_FECI2C)							

**Figure 12-34. Port FECI2C Clear Output Data Register (PCLRR\_FECI2C)**

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0
W					PCLRR_BS			
Reset	0	0	0	0	0	0	0	0
Address	IPSBAR + 0x10_0068 (PCLRR_BS)							

**Figure 12-35. Port BS Clear Output Data Register (PCLRR\_BS)**

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0
W	PCLRR_IRQ							
Reset	0	0	0	0	0	0	0	0
Address	IPSBAR + 0x10_0069 (PCLRR_IRQ)							

Figure 12-36. Port IRQ Clear Output Data Register (PCLRR\_IRQ)

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0
W								PPDSDR_USBH
Reset	0	0	0	0	0	0	0	0
Address	IPSBAR + 0x10_006A (PCLRR_USBH)							

Figure 12-37. Port USBH Clear Output Data Register (PCLRR\_USBH)

Table 12-7. PCLRR\_x Field Descriptions

Name	Description
—	Reserved, should be cleared.
PCLRR_x	Port x clear Data Bits. 0 Always returned for reads; clears corresponding PODR_x bit for writes 1 Never returned for reads; no effect for writes

**Note:** See above figures for bit field positions.

### 12.3.1.5 Pin Assignment Registers (PAR\_x)

The pin assignment registers control which functions are currently active on the external pins. All pin assignment registers are read/write.

#### 12.3.1.5.1 Address Pin Assignment Register (PAR\_AD)

The PAR\_AD register controls the functions of the A[23:21] pins.

	7	6	5	4	3	2	1	0
R	PAR_ADDR23	PAR_ADDR22	PAR_ADDR21	0	0	0	0	0
W								
Reset	See Note			0	0	0	0	0
Address	IPSBAR + 0x10_0070							

**Note:** Reset state determined during reset configuration as shown in [Table 12-8](#).

**Figure 12-38. Address/Data Pin Assignment Register (PAR\_AD)**

**Table 12-8. Reset Values for PAR\_AD Bits**

Mode of Operation	Port Size of External Boot Device	PAR_ADDR23 Reset Value	PAR_ADDR22 Reset Value	PAR_ADDR21 Reset Value
Master mode	8-bit	1	1	1
	16-bit	1	1	1

**Table 12-9. PAR\_AD Field Descriptions**

Bits	Name	Description
7–5	PAR_ADDR	The PAR_ADDR bits configure each of the A[23:21] pins for one of their primary functions or GPIO. 0 A[23:21] pin configured for GPIO 1 A[23:21] pin configured for address bit 23–21 function or $\overline{CS}[6:4]$ function <sup>1</sup>
4–0	—	Reserved, should be cleared.

<sup>1</sup> The selection between the address function and chip select function on each of the A[23:21] pins is determined by the value of the RCSC field in the CIM reset configuration register.

### 12.3.1.5.2 External Bus Control Pin Assignment Register (PAR\_BUSCTL)

The PAR\_BUSCTL register controls the functions of the external bus control signal pins.

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	PAR_OE	PAR_TA	PAR_TEA	PAR_RWB	0	PAR_TSIZ1	PAR_TSIZ0	PAR_TS	PAR_TIP					
W																
Reset	0	0	1	1	1	1	1	0	1	1	1	1	1	1	1	1
Address	IPSBAR + 0x10_0072															

**Figure 12-39. External Bus Control Pin Assignment (PAR\_BUSCTL)**

**Table 12-10. PAR\_BUSCTL Field Descriptions**

Bits	Name	Description																									
15–14	—	Reserved, should be cleared.																									
13	PAR_OE	$\overline{OE}$ pin assignment. The PAR_OE bit configures the $\overline{OE}$ pin for its primary function or GPIO. 0 $\overline{OE}$ pin configured for GPIO 1 $\overline{OE}$ pin configured for external bus OE function																									
12	PAR_TA	$\overline{TA}$ pin assignment. The PAR_TA bit configures the $\overline{OE}$ pin for its primary function or GPIO. 0 $\overline{TA}$ pin configured for GPIO 1 $\overline{TA}$ pin configured for external bus TA function																									
11–10	PAR_TEA	$\overline{TEA}$ pin assignment. The PAR_TEA field configures the $\overline{TEA}$ pin for its primary functions or GPIO. 0x $\overline{TEA}$ pin configured for general purpose I/O 10 $\overline{TEA}$ pin configured for DMA request 1 (DREQ1) function 11 $\overline{TEA}$ pin configured for external bus TEA function																									
9	PAR_RWB	R/ $\overline{W}$ Pin Assignment Bit The PAR_RWB bit configures the R/ $\overline{W}$ pin for its primary function or GPIO. 0 R/ $\overline{W}$ pin configured for GPIO 1 R/ $\overline{W}$ pin configured for external bus read/write function																									
8	—	Reserved, should be cleared																									
7–0	PAR_TSIZ1 PAR_TSIZ0 PAR_TS PAR_TIP	TSIZ[1:0], $\overline{TS}$ , and $\overline{TIP}$ pin assignment. These bit fields configure the TSIZ[1:0], $\overline{TS}$ , and $\overline{TIP}$ pins for one of their primary functions or GPIO. <table><tr><th></th><th>PAR_TSIZ1</th><th>PAR_TSIZ0</th><th>PAR_TS</th><th>PAR_TIP</th></tr><tr><td>00</td><td>GPIO</td><td>GPIO</td><td>GPIO</td><td>GPIO</td></tr><tr><td>01</td><td>GPIO</td><td>GPIO</td><td>GPIO</td><td>GPIO</td></tr><tr><td>10</td><td>DACK1</td><td>DACK0</td><td>DACK2</td><td>DREQ0</td></tr><tr><td>11</td><td>TSIZ1</td><td>TSIZ0</td><td><math>\overline{TS}</math></td><td><math>\overline{TIP}</math></td></tr></table>		PAR_TSIZ1	PAR_TSIZ0	PAR_TS	PAR_TIP	00	GPIO	GPIO	GPIO	GPIO	01	GPIO	GPIO	GPIO	GPIO	10	DACK1	DACK0	DACK2	DREQ0	11	TSIZ1	TSIZ0	$\overline{TS}$	$\overline{TIP}$
	PAR_TSIZ1	PAR_TSIZ0	PAR_TS	PAR_TIP																							
00	GPIO	GPIO	GPIO	GPIO																							
01	GPIO	GPIO	GPIO	GPIO																							
10	DACK1	DACK0	DACK2	DREQ0																							
11	TSIZ1	TSIZ0	$\overline{TS}$	$\overline{TIP}$																							

**12.3.1.5.3 External IRQ Pin Assignment Register (PAR\_IRQ)**

The PAR\_IRQ register controls the functions of the external IRQ<sub>n</sub> pins. To configure these pins for GPIO, set the corresponding bits and refer to [Chapter 14, “Edge Port Module \(EPORT\).”](#)

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	PAR_IRQ7	PAR_IRQ6	PAR_IRQ5	PAR_IRQ4		PAR_IRQ3	PAR_IRQ2	PAR_IRQ1			0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Address	IPSBAR + 0x10_0074															

**Figure 12-40. IRQ Pin Assignment Register (PAR\_IRQ)**

**Table 12-11. PAR\_IRQ Field Descriptions**

Bits	Name	Description																				
15–11	—	Reserved, should be cleared.																				
10–8	PAR_IRQ7 PAR_IRQ6 PAR_IRQ5	$\overline{\text{IRQ}}[7:5]$ pin assignment. Configures the $\overline{\text{IRQ}}[7:5]$ pins for their primary function or GPIO 0 $\overline{\text{IRQ}}[7:5]$ configured for GPIO 1 $\overline{\text{IRQ}}[7:5]$ configured for $\overline{\text{IRQ}}[7:5]$ function																				
7–2	PAR_IRQ4 PAR_IRQ3 PAR_IRQ2	$\overline{\text{IRQ}}[4:2]$ pin assignment. The PAR_IRQ[7:2] bits configure the $\overline{\text{IRQ}}[4:2]$ pins for their primary or alternate functions. <table><tr><td></td><td>PAR_IRQ4</td><td>PAR_IRQ3</td><td>PAR_IRQ2</td></tr><tr><td>00</td><td>Reserved</td><td>Reserved</td><td>Reserved</td></tr><tr><td>01</td><td>Reserved</td><td>Reserved</td><td>Reserved</td></tr><tr><td>10</td><td><math>\overline{\text{DREQ}}2</math></td><td><math>\overline{\text{DREQ}}3</math></td><td><math>\overline{\text{DREQ}}2</math></td></tr><tr><td>11</td><td><math>\overline{\text{IRQ}}4</math></td><td><math>\overline{\text{IRQ}}3</math></td><td><math>\overline{\text{IRQ}}2</math></td></tr></table>		PAR_IRQ4	PAR_IRQ3	PAR_IRQ2	00	Reserved	Reserved	Reserved	01	Reserved	Reserved	Reserved	10	$\overline{\text{DREQ}}2$	$\overline{\text{DREQ}}3$	$\overline{\text{DREQ}}2$	11	$\overline{\text{IRQ}}4$	$\overline{\text{IRQ}}3$	$\overline{\text{IRQ}}2$
	PAR_IRQ4	PAR_IRQ3	PAR_IRQ2																			
00	Reserved	Reserved	Reserved																			
01	Reserved	Reserved	Reserved																			
10	$\overline{\text{DREQ}}2$	$\overline{\text{DREQ}}3$	$\overline{\text{DREQ}}2$																			
11	$\overline{\text{IRQ}}4$	$\overline{\text{IRQ}}3$	$\overline{\text{IRQ}}2$																			
1	PAR_IRQ1	$\overline{\text{IRQ}}1$ pin assignment. Configures the $\overline{\text{IRQ}}1$ pin for primary function or GPIO. 0 $\overline{\text{IRQ}}1$ configured for GPIO 1 $\overline{\text{IRQ}}1$ configured for $\overline{\text{IRQ}}1$ function																				
0	—	Reserved, should be cleared.																				

**12.3.1.5.4 Byte Strobe Pin Assignment Register (PAR\_BS)**

The PAR\_BS register controls the functions of the byte strobe pins.

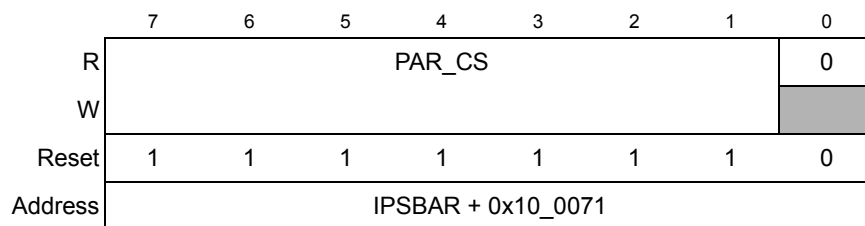
	7	6	5	4	3	2	1	0
R	0	0	0	0	PAR_BS			
W								
Reset	0	0	0	0	1	1	0	0
Address	IPSBAR + 0x10_0084							

**Figure 12-41. Byte Strobe Pin Assignment Register (PAR\_BS)****Table 12-12. PAR\_BS Field Descriptions**

Bits	Name	Description
7–4	—	Reserved, should be cleared.
3–2	PAR_BS	$\overline{\text{BS}}[3:2]$ pin assignment. The PAR_BS[3:2] bits configure the $\overline{\text{BS}}[3:2]$ pins for their primary function or GPIO. 0 $\overline{\text{BS}}[3:2]$ pin configured for GPIO 1 $\overline{\text{BS}}[3:2]$ pin configured for $\overline{\text{BS}}[3:2]$ function Refer to <a href="#">Chapter 9, “Chip Configuration Module (CCM)”</a> for more information on reset configuration.
1–0	—	Reserved, should be cleared.

### 12.3.1.5.5 Chip Select Pin Assignment Register (PAR\_CS)

The PAR\_CS register controls the functions of the EIM chip select pins.



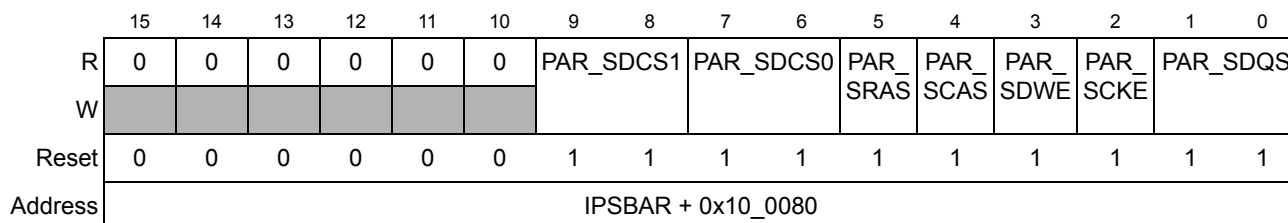
**Figure 12-42. Chip Select Pin Assignment Register (PAR\_CS)**

**Table 12-13. PAR\_CS Field Descriptions**

Bits	Name	Description
7–1	PAR_CS	CS[7:1] pin assignment. The PAR_CS[7:1] bits configure the CS[7:1] pins for their primary functions or GPIO. 0 CS[7:1] pins configured for GPIO 1 CS[7:1] pins configured for EIM CS[7:1] function
0	—	Reserved, should be cleared.

### 12.3.1.5.6 SDRAM Control Pin Assignment Register (PAR\_SDRAM)

The PAR\_SDRAM register controls the function of the SDRAM controller pins.



**Figure 12-43. SDRAM Control Pin Assignment (PAR\_SDRAM)**

**Table 12-14. PAR\_SDRAM Field Descriptions**

Bits	Name	Description															
15–10	—	Reserved, should be cleared.															
9–6	PAR_SDCS1 PAR_SDCS0	<p><math>\overline{\text{SD\_CS}}[1:0]</math> pin assignment. These bits configure the <math>\overline{\text{SD\_CS}}[1:0]</math> pins for their primary functions or GPIO.</p> <table border="1"> <thead> <tr> <th></th><th>PAR_SDCS1</th><th>PAR_SDCS0</th></tr> </thead> <tbody> <tr> <td>00</td><td>GPIO</td><td>GPIO</td></tr> <tr> <td>01</td><td>GPIO</td><td>GPIO</td></tr> <tr> <td>10</td><td><math>\overline{\text{CS3}}</math></td><td><math>\overline{\text{CS2}}</math></td></tr> <tr> <td>11</td><td><math>\overline{\text{SD\_CS1}}</math></td><td><math>\overline{\text{SD\_CS0}}</math></td></tr> </tbody> </table>		PAR_SDCS1	PAR_SDCS0	00	GPIO	GPIO	01	GPIO	GPIO	10	$\overline{\text{CS3}}$	$\overline{\text{CS2}}$	11	$\overline{\text{SD\_CS1}}$	$\overline{\text{SD\_CS0}}$
	PAR_SDCS1	PAR_SDCS0															
00	GPIO	GPIO															
01	GPIO	GPIO															
10	$\overline{\text{CS3}}$	$\overline{\text{CS2}}$															
11	$\overline{\text{SD\_CS1}}$	$\overline{\text{SD\_CS0}}$															
5	PAR_SRAS	<p><math>\overline{\text{SD\_SRAS}}</math> pin assignment. This bit configures the <math>\overline{\text{SD\_SRAS}}</math> pin for its primary function or GPIO.</p> <p>0 <math>\overline{\text{SD\_SRAS}}</math> pin configured for GPIO 1 <math>\overline{\text{SD\_SRAS}}</math> pin configured for SDRAMC RAS function</p>															
4	PAR_SCAS	<p><math>\overline{\text{SD\_SCAS}}</math> pin assignment. This bit configures the <math>\overline{\text{SD\_SCAS}}</math> pin for its primary function or GPIO.</p> <p>0 <math>\overline{\text{SD\_SCAS}}</math> pin configured for GPIO 1 <math>\overline{\text{SD\_SCAS}}</math> pin configured for SDRAMC CAS function</p>															
3	PAR_SDWE	<p><math>\overline{\text{SD\_WE}}</math> pin assignment. This bit configures the <math>\overline{\text{SD\_WE}}</math> pin for its primary function or GPIO.</p> <p>0 <math>\overline{\text{SD\_WE}}</math> pin configured for GPIO 1 <math>\overline{\text{SD\_WE}}</math> pin configured for SDRAMC WE function</p>															
2	PAR_SCKE	<p><math>\overline{\text{SD\_CKE}}</math> pin assignment. This bit configures the <math>\overline{\text{SD\_CKE}}</math> pin for its primary function or GPIO.</p> <p>0 <math>\overline{\text{SD\_CKE}}</math> pin configured for GPIO 1 <math>\overline{\text{SD\_CKE}}</math> pin configured for SDRAMC clock enable function</p>															
1–0	PAR_SDQS	<p><math>\overline{\text{SD\_DQS}}[3:2]</math> pin assignment. These bits configure the <math>\overline{\text{SD\_DQS}}[3:2]</math> pins for their primary functions or GPIO.</p> <p>0 <math>\overline{\text{SD\_DQS}}[3:2]</math> pin configured for GPIO 1 <math>\overline{\text{SD\_DQS}}[3:2]</math> pin configured for <math>\overline{\text{SD\_DQS}}[3:2]</math> function</p>															

**12.3.1.5.7 FEC/I2C Pin Assignment Register (PAR\_FECI2C)**

The PAR\_FECI2C register controls the functions of the I<sup>2</sup>C and some of the FEC pins.

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	PAR_MDIO0		PAR_MDC0		PAR_MDIO1	0	PAR_MDC1	0	PAR_SDA		PAR_SCL	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Address	IPSBAR + 0x10_0082															

**Figure 12-44. FEC/I2C Pin Assignment (PAR\_FECI2C)**



**Table 12-15. PAR\_FECI2C Field Descriptions**

Bits	Name	Description															
15–12	—	Reserved, should be cleared.															
11–8	PAR_MDIO0 PAR_MDC0	<p>FEC 0 pin assignment. These bit fields configure the FEC0_MDIO and FEC0_MDC pins for one of their primary functions or GPIO.</p> <table border="1"> <thead> <tr> <th></th><th>PAR_MDIO0</th><th>PAR_MDC0</th></tr> </thead> <tbody> <tr> <td>00</td><td>GPIO</td><td>GPIO</td></tr> <tr> <td>01</td><td>U2RXD</td><td>U2TXD</td></tr> <tr> <td>10</td><td>I2C_SDA</td><td>I2C_SCL</td></tr> <tr> <td>11</td><td>FEC0_MDIO</td><td>FEC0_MDC</td></tr> </tbody> </table>		PAR_MDIO0	PAR_MDC0	00	GPIO	GPIO	01	U2RXD	U2TXD	10	I2C_SDA	I2C_SCL	11	FEC0_MDIO	FEC0_MDC
	PAR_MDIO0	PAR_MDC0															
00	GPIO	GPIO															
01	U2RXD	U2TXD															
10	I2C_SDA	I2C_SCL															
11	FEC0_MDIO	FEC0_MDC															
7	PAR_MDIO1	<p>FEC1_MDIO pin assignment. This bit configures the FEC1_MDIO pin for its primary function or GPIO.</p> <p>0 FEC1_MDIO pin configured for GPIO 1 FEC1_MDIO pin configured for FEC MDIO function</p>															
6	—	Reserved, should be cleared.															
5	PAR_MDC1	<p>FEC1_MDC pin assignment. This bit configures the FEC1_MDC pin for its primary function or GPIO.</p> <p>0 FEC1_MDC pin configured for GPIO 1 FEC1_MDC pin configured for FEC MDC function</p>															
4	—	Reserved, should be cleared.															
3–0	PAR_SDA PAR_SCL	<p>I<sup>2</sup>C pin assignment. These bit fields configure the I2C_SDA and I2C_SCL pins for one of their primary functions or GPIO.</p> <table border="1"> <thead> <tr> <th></th><th>PAR_SDA</th><th>PAR_SCL</th></tr> </thead> <tbody> <tr> <td>00</td><td>GPIO</td><td>GPIO</td></tr> <tr> <td>01</td><td>U2RXD</td><td>U2TXD</td></tr> <tr> <td>10</td><td>GPIO</td><td>GPIO</td></tr> <tr> <td>11</td><td>I2C_SDA</td><td>I2C_SCL</td></tr> </tbody> </table>		PAR_SDA	PAR_SCL	00	GPIO	GPIO	01	U2RXD	U2TXD	10	GPIO	GPIO	11	I2C_SDA	I2C_SCL
	PAR_SDA	PAR_SCL															
00	GPIO	GPIO															
01	U2RXD	U2TXD															
10	GPIO	GPIO															
11	I2C_SDA	I2C_SCL															

**12.3.1.5.8 UART Pin Assignment Register (PAR\_UART)**

The PAR\_UART register controls the functions of the UART pins.

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	PAR_U2RXD	PAR_U2TXD	PAR_U2CTS	PAR_U2RTS	PAR_U1RXD	PAR_U1TXD	PAR_U1CTL	PAR_U0RXD	PAR_U0TXD	PAR_U0CTL				
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Address	IPSBAR + 0x10_007C															

Figure 12-45. UART Pin Assignment (PAR\_UART)

Table 12-16. PAR\_UART Field Descriptions

Bits	Name	Description															
15–14	—	Reserved, should be cleared.															
13	PAR_U2RXD	U2RXD pin assignment. This bit configures the U2RXD pin for its primary function or GPIO. 0 U2RXD pin configured for GPIO 1 U2RXD pin configured for UART2 receive data function															
12	PAR_U2TXD	U2TXD pin assignment. This bit configures the U2TXD pin for its primary function or GPIO. 0 U2TXD pin configured for GPIO 1 U2TXD pin configured for UART2 transmit data function															
11–8	PAR_U2CTS PAR_U2RTS	UART 2 pin assignment. These bit fields configure the U2CTS, and U2RTS pins for one of their primary functions or GPIO. <table border="1"> <thead> <tr> <th></th><th>PAR_U2CTS</th><th>PAR_U2RTS</th></tr> </thead> <tbody> <tr> <td>00</td><td>GPIO</td><td>GPIO</td></tr> <tr> <td>01</td><td>GPIO</td><td>GPIO</td></tr> <tr> <td>10</td><td>PWM1</td><td>PWM0</td></tr> <tr> <td>11</td><td>U2CTS</td><td>U2RTS</td></tr> </tbody> </table>		PAR_U2CTS	PAR_U2RTS	00	GPIO	GPIO	01	GPIO	GPIO	10	PWM1	PWM0	11	U2CTS	U2RTS
	PAR_U2CTS	PAR_U2RTS															
00	GPIO	GPIO															
01	GPIO	GPIO															
10	PWM1	PWM0															
11	U2CTS	U2RTS															
7–6	PAR_U1RXD PAR_U1TXD	UART 1 pin assignment. These bit fields configure the U1RXD and U1TXD pins for their primary function or GPIO. <table border="1"> <thead> <tr> <th></th><th>PAR_U1RXD</th><th>PAR_U1TXD</th></tr> </thead> <tbody> <tr> <td>0</td><td>GPIO</td><td>GPIO</td></tr> <tr> <td>1</td><td>U1RXD</td><td>U1TXD</td></tr> </tbody> </table>		PAR_U1RXD	PAR_U1TXD	0	GPIO	GPIO	1	U1RXD	U1TXD						
	PAR_U1RXD	PAR_U1TXD															
0	GPIO	GPIO															
1	U1RXD	U1TXD															
5–4	PAR_U1CTL	U1CTS and U1RTS pin assignment. These two bits configure the U1CTS and U1RTS pins for primary function or GPIO. 00 U1CTS and U1RTS pins configured for GPIO 01 Reserved 10 Reserved 11 U1CTS and U1RTS pins configured for UART1 control functions.															

**Table 12-16. PAR\_UART Field Descriptions (Continued)**

Bits	Name	Description									
3–2	PAR_U0RXD PAR_U0TXD	UART 0 pin assignment. These bit fields configure the U0RXD and U0TXD pins for their primary function or GPIO. <div> <table> <tr> <th></th><th>PAR_U0RXD</th><th>PAR_U0TXD</th></tr> <tr> <td>0</td><td>GPIO</td><td>GPIO</td></tr> <tr> <td>1</td><td>U0RXD</td><td>U0TXD</td></tr> </table> </div>		PAR_U0RXD	PAR_U0TXD	0	GPIO	GPIO	1	U0RXD	U0TXD
	PAR_U0RXD	PAR_U0TXD									
0	GPIO	GPIO									
1	U0RXD	U0TXD									
1-0	PAR_U0CTL	U0CTS and U0RTS pin assignment. These two bits configure the U0CTS and U0RTS pins for primary function or GPIO. 00 U0CTS and U0RTS pins configured for GPIO 01 Reserved 10 Reserved 11 U0CTS and U0RTS pins configured for UART0 control functions.									

**12.3.1.5.9 QSPI Pin Assignment Register (PAR\_QSPI)**

The PAR\_QSPI register controls the functions of the QSPI pins.

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	PAR_PCS3	PAR_PCS2	PAR_PCS1	0	PAR_PCS0	0	PAR_SCK	PAR_DIN	PAR_DOUT	0				
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Address	IPSBAR + 0x10_007E															

**Figure 12-46. QSPI Pin Assignment Register (PAR\_QSPI)****Table 12-17. PAR\_QSPI Field Descriptions**

Bits	Name	Description															
15–14	—	Reserved, should be cleared.															
13–10	PAR_PCS3 PAR_PCS2	QSPI_PCS[3:2] pin assignment. These fields configure the QSPI_PCS[3:2] pins for one of their primary functions or GPIO. <div> <table> <tr> <th></th><th>PAR_PCS3</th><th>PAR_PCS2</th></tr> <tr> <td>00</td><td>GPIO</td><td>GPIO</td></tr> <tr> <td>01</td><td>DACK3</td><td>DACK2</td></tr> <tr> <td>10</td><td>PWM3</td><td>PWM2</td></tr> <tr> <td>11</td><td>QSPI_PCS3</td><td>QSPI_PCS2</td></tr> </table> </div>		PAR_PCS3	PAR_PCS2	00	GPIO	GPIO	01	DACK3	DACK2	10	PWM3	PWM2	11	QSPI_PCS3	QSPI_PCS2
	PAR_PCS3	PAR_PCS2															
00	GPIO	GPIO															
01	DACK3	DACK2															
10	PWM3	PWM2															
11	QSPI_PCS3	QSPI_PCS2															

**Table 12-17. PAR\_QSPI Field Descriptions (Continued)**

Bits	Name	Description															
9	PAR_PCS1	QSPI_PCS1 pin assignment. This bit configures the QSPI_PCS1 pin for its primary function or GPIO. 0 QSPI_PSC1 pin configured for GPIO 1 QSPI_PSC1 pin configured for QSPI PCS1 function															
8	—	Reserved, should be cleared.															
7	PAR_PCS0	QSPI_PCS0 pin assignment. This bit configures the QSPI_PCS0 pin for its primary function or GPIO. 0 QSPI_PSC0 pin configured for GPIO 1 QSPI_PSC0 pin configured for QSPI PCS0 function															
6	—	Reserved, should be cleared.															
5–2	PAR_SCK PAR_DIN	QSPI_SCK & QSPI_DIN pin assignment. These fields configure the QSPI_SCK & QSPI_DIN pins for one of their primary functions or GPIO. <table border="1"> <thead> <tr> <th></th><th>PAR_SCK</th><th>PAR_DIN</th></tr> </thead> <tbody> <tr> <td>00</td><td>GPIO</td><td>GPIO</td></tr> <tr> <td>01</td><td>GPIO</td><td>GPIO</td></tr> <tr> <td>10</td><td>I2C_SCL</td><td>I2C_SDA</td></tr> <tr> <td>11</td><td>QSPI_SCK</td><td>QSPI_DIN</td></tr> </tbody> </table>		PAR_SCK	PAR_DIN	00	GPIO	GPIO	01	GPIO	GPIO	10	I2C_SCL	I2C_SDA	11	QSPI_SCK	QSPI_DIN
	PAR_SCK	PAR_DIN															
00	GPIO	GPIO															
01	GPIO	GPIO															
10	I2C_SCL	I2C_SDA															
11	QSPI_SCK	QSPI_DIN															
1	PAR_DOUT	QSPI_DOUT pin assignment. This bit configures the QSPI_DOUT pin for its primary function or GPIO. 0 QSPI_DOUT pin configured for GPIO 1 QSPI_DOUT pin configured for QSPI DOUT function															
0	—	Reserved, should be cleared.															

### 12.3.1.6 Timer Pin Assignment Registers (PAR\_TIMERH & PAR\_TIMERL)

The PAR\_TIMERH and PAR\_TIMERL registers control the functions of the DMA timer pins.

	7	6	5	4	3	2	1	0
R	PAR_T3IN		PAR_T3OUT		PAR_T2IN		PAR_T2OUT	
W								
Reset	0	0	0	0	0	0	0	0
Address	IPSBAR + 0x10_007A							

**Figure 12-47. Timer Pin Assignment Register (PAR\_TIMERH)**

**Table 12-18. PAR\_TIMERH Field Description**

Bits	Name	Description															
7–4	PAR_T3IN PAR_T3OUT	<p>Timer 3 pin assignment. These bit fields configure the T3IN and T3OUT pins for one of their primary functions or GPIO.</p> <table> <tr> <th></th><th>PAR_T3IN</th><th>PAR_T3OUT</th></tr> <tr> <td>00</td><td>GPIO</td><td>GPIO</td></tr> <tr> <td>01</td><td><math>\overline{\text{U2RTS}}</math></td><td><math>\overline{\text{U2CTS}}</math></td></tr> <tr> <td>10</td><td>T3OUT</td><td>PWM3</td></tr> <tr> <td>11</td><td>T3IN</td><td>T3OUT</td></tr> </table>		PAR_T3IN	PAR_T3OUT	00	GPIO	GPIO	01	$\overline{\text{U2RTS}}$	$\overline{\text{U2CTS}}$	10	T3OUT	PWM3	11	T3IN	T3OUT
	PAR_T3IN	PAR_T3OUT															
00	GPIO	GPIO															
01	$\overline{\text{U2RTS}}$	$\overline{\text{U2CTS}}$															
10	T3OUT	PWM3															
11	T3IN	T3OUT															
3–0	PAR_T2IN PAR_T2OUT	<p>Timer 2 pin assignment. These bit fields configure the T2IN and T2OUT pins for one of their primary functions or GPIO.</p> <table> <tr> <th></th><th>PAR_T2IN</th><th>PAR_T2OUT</th></tr> <tr> <td>00</td><td>GPIO</td><td>GPIO</td></tr> <tr> <td>01</td><td>GPIO</td><td>GPIO</td></tr> <tr> <td>10</td><td>T2OUT</td><td>PWM2</td></tr> <tr> <td>11</td><td>T2IN</td><td>T2OUT</td></tr> </table>		PAR_T2IN	PAR_T2OUT	00	GPIO	GPIO	01	GPIO	GPIO	10	T2OUT	PWM2	11	T2IN	T2OUT
	PAR_T2IN	PAR_T2OUT															
00	GPIO	GPIO															
01	GPIO	GPIO															
10	T2OUT	PWM2															
11	T2IN	T2OUT															

	7	6	5	4	3	2	1	0
R	PAR_T1IN		PAR_T1OUT		PAR_T0IN		PAR_T0OUT	
W								
Reset	0	0	0	0	0	0	0	0
Address	IPSBAR + 0x10_007B							

**Figure 12-48. Timer Pin Assignment Register (PAR\_TIMERL)**

**Table 12-19. PAR\_TIMERL Field Description**

Bits	Name	Description															
7–4	PAR_T1IN PAR_T1OUT	Timer 1 pin assignment. These bit fields configure the T1IN and T1OUT pins for one of their primary functions or GPIO. <table border="1"> <thead> <tr> <th></th><th>PAR_T1IN</th><th>PAR_T1OUT</th></tr> </thead> <tbody> <tr> <td>00</td><td>GPIO</td><td>GPIO</td></tr> <tr> <td>01</td><td>GPIO</td><td>GPIO</td></tr> <tr> <td>10</td><td>T1OUT</td><td>PWM1</td></tr> <tr> <td>11</td><td>T1IN</td><td>T1OUT</td></tr> </tbody> </table>		PAR_T1IN	PAR_T1OUT	00	GPIO	GPIO	01	GPIO	GPIO	10	T1OUT	PWM1	11	T1IN	T1OUT
	PAR_T1IN	PAR_T1OUT															
00	GPIO	GPIO															
01	GPIO	GPIO															
10	T1OUT	PWM1															
11	T1IN	T1OUT															
3–0	PAR_T0IN PAR_T0OUT	Timer 1 pin assignment. These bit fields configure the T0IN and T0OUT pins for one of their primary functions or GPIO. <table border="1"> <thead> <tr> <th></th><th>PAR_T0IN</th><th>PAR_T0OUT</th></tr> </thead> <tbody> <tr> <td>00</td><td>GPIO</td><td>GPIO</td></tr> <tr> <td>01</td><td>GPIO</td><td>GPIO</td></tr> <tr> <td>10</td><td>T0OUT</td><td>PWM0</td></tr> <tr> <td>11</td><td>T0IN</td><td>T0OUT</td></tr> </tbody> </table>		PAR_T0IN	PAR_T0OUT	00	GPIO	GPIO	01	GPIO	GPIO	10	T0OUT	PWM0	11	T0IN	T0OUT
	PAR_T0IN	PAR_T0OUT															
00	GPIO	GPIO															
01	GPIO	GPIO															
10	T0OUT	PWM0															
11	T0IN	T0OUT															

### 12.3.1.7 USB Pin Assignment Register (PAR\_USB)

The PAR\_USB register controls the functions of the USB pins.

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	PAR_USPD	PAR_UCLK	PAR_URN	PAR_URP	PAR_URX	PAR_USSP	PAR_UTN	PAR_UTP	PAR_UTXEN
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Address	IPSBAR + 0x10_0076															

**Figure 12-49. USB Pin Assignment Register (PAR\_USB)****Table 12-21. PAR\_USB Descriptions**

Bits	Name	Description
15–9	—	Reserved, should be cleared.
8	PAR_USPD	USB_SPEED pin assignment. This bit configures the USB_SPEED pin for its primary function or GPIO. 0 USB_SPEED pin configured for GPIO 1 USB_SPEED pin configured for USB SPEED function

**Table 12-21. PAR\_USB Descriptions (Continued)**

Bits	Name	Description
7	PAR_UCLK	USB_CLK pin assignment. This bit configures the USB_CLK pin for its primary function or GPIO. 0 USB_CLK pin configured for GPIO 1 USB_CLK pin configured for USB CLK function
6	PAR_URN	USB_RN pin assignment. This bit configures the USB_RN pin for its primary function or GPIO. 0 USB_RN pin configured for GPIO 1 USB_RN pin configured for USB RN function
5	PAR_URP	USB_RP pin assignment. This bit configures the USB_RP pin for its primary function or GPIO. 0 USB_RP pin configured for GPIO 1 USB_RP pin configured for USB RP function
4	PAR_URX	USB_RXD pin assignment. This bit configures the USB_RXD pin for its primary function or GPIO. 0 USB_RXD pin configured for GPIO 1 USB_RXD pin configured for USB RX function
3	PAR_USSP	USB_SUSP pin assignment. This bit configures the USB_SUSP pin for its primary function or GPIO. 0 USB_SUSP pin configured for GPIO 1 USB_SUSP pin configured for USB SUSP function
2	PAR_UTN	USB_TN pin assignment. This bit configures the USB_TN pin for its primary function or GPIO. 0 USB_TN pin configured for GPIO 1 USB_TN pin configured for USB TN function
1	PAR_UTC	USB_TP pin assignment. This bit configures the USB_TP pin for its primary function or GPIO. 0 USB_TP pin configured for GPIO 1 USB_TP pin configured for USB TP function
0	PAR_UTCEN	USB_TXEN pin assignment. This bit configures the USB_TXEN pin for its primary function or GPIO. 0 USB_TXEN pin configured for GPIO 1 USB_TXEN pin configured for USB TX EN function

### 12.3.1.8 FEC0 & FEC1 Pin Assignment Registers (PAR\_FEC0 & PAR\_FEC1)

The PAR\_FEC0 and PAR\_FEC1 registers control the functions of the FEC pins.

	7	6	5	4	3	2	1	0
R	PAR_FEC0H	PAR_FEC0L	0	0	0	0	0	0
W								
Reset	0	0	0	0	0	0	0	0
Address	IPSBAR + 0x10_0078							

**Figure 12-50. FEC Pin Assignment Register (PAR\_FEC0)**

	7	6	5	4	3	2	1	0
R	PAR_FEC1H	PAR_FEC1L	0	0	0	0	0	0
W								
Reset	0	0	0	0	0	0	0	0
Address	IPSBAR + 0x10_0079							

**Figure 12-51. FEC Pin Assignment Register (PAR\_FEC1)****Table 12-22. PAR\_FEC0 & PAR\_FEC1 Field Descriptions**

Bits	Name	Description
7	PAR_FEC0H PAR_FEC1H	FECnH pin assignment. These bits configure the FECnH pins for their primary function or GPIO. 0 FECnH pins configured for GPIO 1 FECnH pins configured for ethernet 7-wire mode
6	PAR_FEC0L PAR_FEC1L	FECnL pin assignment. These bits configure the FECnL pins for their primary function or GPIO. 0 FECnL pins configured for GPIO 1 FECnL pins configured for ethernet full mode
5–0	—	Reserved, should be cleared.

## 12.4 Functional Description

### 12.4.1 Overview

Initial pin function is determined during reset configuration. The pin assignment registers allow the user to select among various primary functions and general purpose I/O after reset.

Most pins are configured as general purpose I/O by default. The notable exceptions to this are external bus control pins, address/data pins, and chip select pins. These pins are configured for their primary functions after reset.

Every general purpose I/O pin is individually configurable as an input or an output via a data direction register (PDDR\_x).

Every GPIO port has an output data register (PODR\_x) and a pin data register (PPDSDR\_x) to monitor and control the state of its pins. Data written to a PODR\_x register is stored and then driven to the corresponding port x pins configured as outputs.

Reading a PODR\_x register returns the current state of the register regardless of the state of the corresponding pins.

Reading a PPDSDR\_x register returns the current state of the corresponding pins when configured as general purpose I/O, regardless of whether the pins are inputs or outputs.



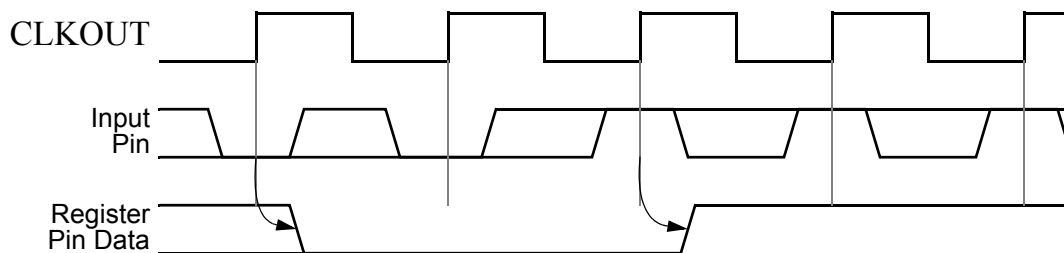
Every GPIO port has a PPDSDR\_ $x$  register and a clear register (PCLRR\_ $x$ ) for setting or clearing individual bits in the PODR\_ $x$  register.

Initial pin output drive strength is determined during reset configuration. The DSCR\_ $x$  registers allow the pin drive strengths to be configured on a per-function basis after reset.

The MCF5275 ports module does not generate interrupt requests.

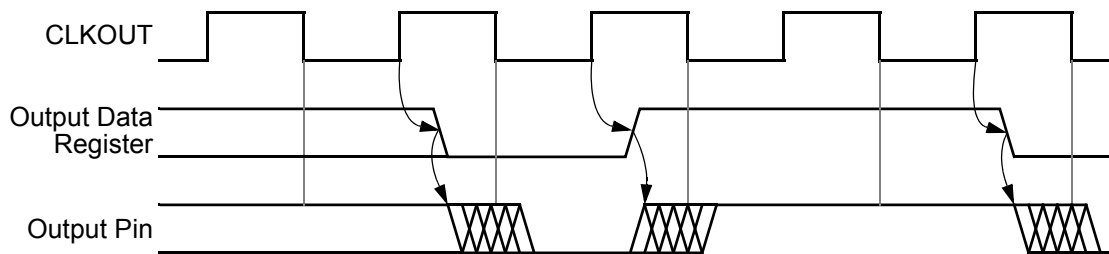
## 12.4.2 Port Digital I/O Timing

Input data on all pins configured as general purpose input is synchronized to the rising edge of the internal bus clock, CLKOUT, as shown in [Figure 12-52](#).



**Figure 12-52. General Purpose Input Timing**

Data written to the PODR\_ $x$  register of any pin configured as a general purpose output is immediately driven to its respective pin, as shown in [Figure 12-53](#).



**Figure 12-53. General Purpose Output Timing**

## 12.5 Initialization/Application Information

The initialization for the MCF5275 ports module is done during reset configuration. All registers are reset to a predetermined state. Refer to [Section 12.3, “Memory Map/Register Definition,”](#) for more details on reset and initialization.



# Chapter 13

## Interrupt Controller Modules

### 13.1 Introduction

This section details the functionality for the MCF5275 interrupt controllers (INTC0, INTC1). The general features of the MCF5275 interrupt controller block include:

- 58 interrupt sources, organized as:
  - 51 fully-programmable interrupt sources
  - 7 fixed-level interrupt sources
- Each of the 58 sources has a unique interrupt control register (ICR $_{nx}$ ) to define the software-assigned levels and priorities within the level
- Unique vector number for each interrupt source
- Ability to mask any individual interrupt source, plus global mask-all capability
- Supports both hardware and software interrupt acknowledge cycles
- “Wake-up” signal from low-power stop modes

The 51 fully-programmable and seven fixed-level interrupt sources for the two interrupt controllers on the MCF5275 handle the complete set of interrupt sources from all of the modules on the device. This section describes how the interrupt sources are mapped to the interrupt controller logic and how interrupts are serviced.

#### 13.1.1 68K/ColdFire Interrupt Architecture Overview

Before continuing with the specifics of the MCF5275 interrupt controllers, a brief review of the interrupt architecture of the 68K/ColdFire family is appropriate.

The interrupt architecture of ColdFire is exactly the same as the M68000 family, where there is a 3-bit encoded interrupt priority level sent from the interrupt controller to the core, providing 7 levels of interrupt requests. Level 7 represents the highest priority interrupt level, while level 1 is the lowest priority. The processor samples for active interrupt requests once per instruction by comparing the encoded priority level against a 3-bit interrupt mask value (I) contained in bits 10:8 of the machine’s status register (SR). If the priority level is greater than the SR[I] field at the sample point, the processor suspends normal instruction execution and initiates interrupt exception processing. Level 7 interrupts are treated as non-maskable and edge-sensitive within the processor, while levels 1-6 are treated as level-sensitive and may be masked depending on the value of the SR[I] field. For correct operation, the ColdFire requires that, once asserted, the interrupt source remain asserted until explicitly disabled by the interrupt service routine.

During the interrupt exception processing, the CPU enters supervisor mode, disables trace mode and then fetches an 8-bit vector from the interrupt controller. This byte-sized operand fetch is known as the interrupt acknowledge (IACK) cycle with the ColdFire implementation using a special encoding of the transfer type and transfer modifier attributes to distinguish this data fetch from a “normal” memory access. The fetched data provides an index into the exception vector table which contains 256 addresses, each pointing to the beginning of a specific exception service routine. In particular, vectors 64 - 255 of the exception vector table are reserved for user interrupt service routines. The first 64 exception vectors are reserved for the processor to handle reset, error conditions (access, address), arithmetic faults, system calls, etc. Once the interrupt vector number has been retrieved, the processor continues by creating a stack frame in memory. For ColdFire, all exception stack frames are 2 longwords in length, and contain 32 bits of vector and status register data, along with the 32-bit program counter value of the instruction that was interrupted (see [Section 3.6, “Exception Stack Frame Definition”](#) for more information on the stack frame format). After the exception stack frame is stored in memory, the processor accesses the 32-bit pointer from the exception vector table using the vector number as the offset, and then jumps to that address to begin execution of the service routine. After the status register is stored in the exception stack frame, the SR[I] mask field is set to the level of the interrupt being acknowledged, effectively masking that level and all lower values while in the service routine. For many peripheral devices, the processing of the IACK cycle directly negates the interrupt request, while other devices require that request to be explicitly negated during the processing of the service routine.

For the MCF5275, the processing of the interrupt acknowledge cycle is fundamentally different than previous 68K/ColdFire cores. In the new approach, all IACK cycles are directly handled by the interrupt controller, so the requesting peripheral device is not accessed during the IACK. As a result, the interrupt request must be explicitly cleared in the peripheral during the interrupt service routine. For more information, see [Section 13.1.2.3, “Interrupt Vector Determination.”](#)

Unlike the M68000 family, all ColdFire processors guarantee that the first instruction of the service routine is executed before sampling for interrupts is resumed. By making this initial instruction a load of the SR, interrupts can be safely disabled, if required.

During the execution of the service routine, the appropriate actions must be performed on the peripheral to negate the interrupt request.

For more information on exception processing, see the *ColdFire Programmer's Reference Manual* at <http://www.freescale.com/coldfire>.

## 13.1.2 Interrupt Controller Theory of Operation

To support the interrupt architecture of the 68K/ColdFire programming model, the combined 63 interrupt sources are organized as 7 levels, with each level supporting up to 9 prioritized requests. Consider the priority structure within a single interrupt level (from highest to lowest priority) as shown in [Table 13-1](#).

**Table 13-1. Interrupt Priority Within a Level**

ICR[2:0]	Priority	Interrupt Sources
111	7 (Highest)	8–63
110	6	8–63
101	5	8–63
100	4	8–63
—	Fixed Midpoint Priority	1–7
011	3	8–63
010	2	8–63
001	1	8–63
000	0 (Lowest)	8–63

The level and priority is fully programmable for all sources except interrupt sources 1–7. Interrupt source 1–7 ( $\overline{\text{IRQ}}[1:7]$  from the Edgeport module) are fixed at the corresponding level's midpoint priority. Thus, a maximum of 8 fully-programmable interrupt sources are mapped into a single interrupt level. The “fixed” interrupt source is hardwired to the given level, and represents the mid-point of the priority within the level. For the fully-programmable interrupt sources, the 3-bit level and the 3-bit priority within the level are defined in the 8-bit interrupt control register ( $\text{ICR}_{nx}$ ).

The operation of the interrupt controller can be broadly partitioned into three activities:

- Recognition
- Prioritization
- Vector determination during IACK

### 13.1.2.1 Interrupt Recognition

The interrupt controller continuously examines the request sources and the interrupt mask register to determine if there are active requests. This is the recognition phase.

### 13.1.2.2 Interrupt Prioritization

As an active request is detected, it is translated into the programmed interrupt level, and the resulting 7-bit decoded priority level ( $\text{IRQ}[7:1]$ ) is driven out of the interrupt controller. The decoded priority levels from all the interrupt controllers are logically summed together and the highest enabled interrupt request is then encoded into a 3-bit priority level that is sent to the processor core during this prioritization phase.

### 13.1.2.3 Interrupt Vector Determination

Once the core has sampled for pending interrupts and begun interrupt exception processing, it generates an interrupt acknowledge cycle (IACK). The IACK transfer is treated as a memory-mapped byte read by the processor, and routed to the appropriate interrupt controller. Next, the interrupt controller extracts the level being acknowledged from address bits[4:2], and then determines the highest priority interrupt request active for that level, and returns the 8-bit interrupt vector for that request to complete the cycle. The 8-bit interrupt vector is formed using the following algorithm:

```
For INTC0,          vector_number = 64 + interrupt source number
For INTC1,          vector_number = 128 + interrupt source number
```

Recall vector\_numbers 0 - 63 are reserved for the ColdFire processor and its internal exceptions. Thus, the following mapping of bit positions to vector numbers applies for the INTC0:

```
if interrupt source 1 is active and acknowledged, then vector_number = 65
if interrupt source 2 is active and acknowledged, then vector_number = 66
...
if interrupt source 8 is active and acknowledged, then vector_number = 72
if interrupt source 9 is active and acknowledged, then vector_number = 73
...
if interrupt source 62 is active and acknowledged, then vector_number = 126
```

The net effect is a fixed mapping between the bit position within the source to the actual interrupt vector number.

If there is no active interrupt source for the given level, a special “spurious interrupt” vector (vector\_number = 24) is returned and it is the responsibility of the service routine to handle this error situation.

Note this protocol implies the interrupting peripheral is not accessed during the acknowledge cycle since the interrupt controller completely services the acknowledge. This means the interrupt source must be explicitly disabled in the interrupt service routine. This design provides unique vector capability for all interrupt requests, regardless of the “complexity” of the peripheral device.

## 13.2 Memory Map/Register Definition

The register programming model for the interrupt controllers is memory-mapped to a 256-byte space. In the following discussion, there are a number of program-visible registers greater than 32 bits in size. For these control fields, the physical register is partitioned into two 32-bit values: a register “high” (the upper longword) and a register “low” (the lower longword). The nomenclature <reg\_name>H and <reg\_name>L is used to reference these values.

The registers and their locations are defined in [Table 13-3](#). The offsets listed start from the base address for each interrupt controller. The base addresses for the interrupt controllers are listed below:

**Table 13-2. Interrupt Controller Base Addresses**

Interrupt Controller Number	Base Address
INTC0	IPSBAR + 0xC00
INTC1	IPSBAR + 0xD00
Global IACK Registers Space <sup>1</sup>	IPSBAR + 0xF00

<sup>1</sup> This address space only contains the SWIACK and L1ACK-L7IACK registers. See [Section 13.2.1.7](#), "Software and Level n IACK Registers (SWIACKR, L1IACK–L7IACK)" for more information

**Table 13-3. Interrupt Controller Memory Map**

IPSBAR Offset	Bits[31:24]	Bits[23:16]	Bits[15:8]	Bits[7:0]
0xC00 & 0xD00	Interrupt Pending Register High (IPRH), [63:32]			
0xC04 & 0xD04	Interrupt Pending Register Low (IPRL), [31:1]			
0xC08 & 0xD08	Interrupt Mask Register High (IMRH), [63:32]			
0xC0C & 0xD0C	Interrupt Mask Register Low (IMRL), [31:0]			
0xC10 & 0xD10	Interrupt Force Register High (INTFRCH), [63:32]			
0xC14 & 0xD14	Interrupt Force Register Low (INTFRCL), [31:1]			
0xC18 & 0xD18	IRLR[7:1]	IACKLPR[7:0]	Reserved	
0xC1C - 0xC3C & 0xD1C - 0xD3C	Reserved			
0xC40 & 0xD40	Reserved	ICR01	ICR02	ICR03
0xC44 & 0xD44	ICR04	ICR05	ICR06	ICR07
0xC48 & 0xD48	ICR08	ICR09	ICR10	ICR11
0xC4C & 0xD4C	ICR12	ICR13	ICR14	ICR15
0xC50 & 0xD50	ICR16	ICR17	ICR18	ICR19
0xC54 & 0xD54	ICR20	ICR21	ICR22	ICR23
0xC58 & 0xD58	ICR24	ICR25	ICR26	ICR27
0xC5C & 0xD5C	ICR28	ICR29	ICR30	ICR31
0xC60 & 0xD60	ICR32	ICR33	ICR34	ICR35
0xC64 & 0xD64	ICR36	ICR37	ICR38	ICR39
0xC68 & 0xD68	ICR40	ICR41	ICR42	ICR43
0xC6C & 0xD6C	ICR44	ICR45	ICR46	ICR47
0xC70 & 0xD70	ICR48	ICR49	ICR50	ICR51
0xC74 & 0xD74	ICR52	ICR53	ICR54	ICR55

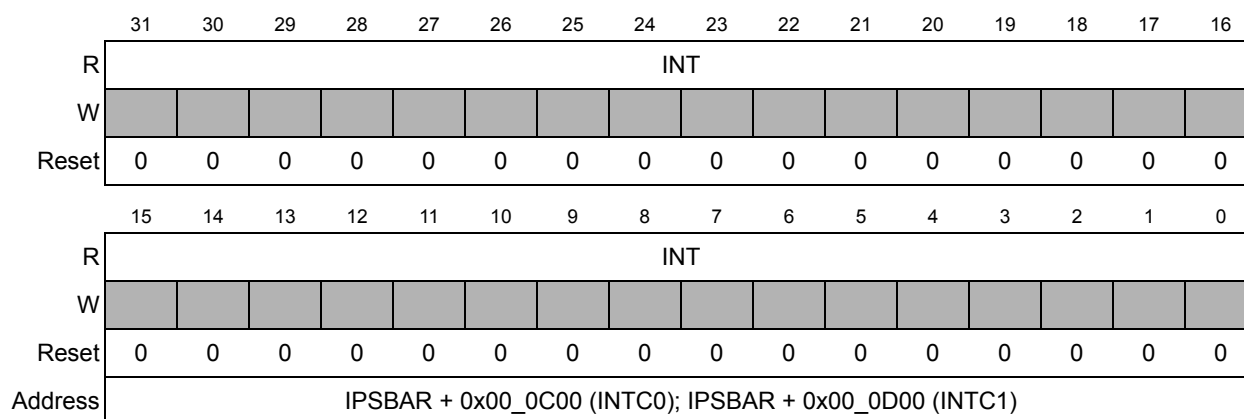
**Table 13-3. Interrupt Controller Memory Map (Continued)**

IPSBAR Offset	Bits[31:24]	Bits[23:16]	Bits[15:8]	Bits[7:0]
0xC78 & 0xD78	ICR56	ICR57	ICR58	ICR59
0xC7C & 0xD7C	ICR60	ICR61	ICR62	ICR63
0xC80 - 0xCDC & 0xD80 - & 0xDDC	Reserved			
0xCE0 & 0xDE0	SWIACK	Reserved		
0xCE4 & 0xDE4	L1IACK	Reserved		
0xCE8 & 0xDE8	L2IACK	Reserved		
0xCEC & 0xDEC	L3IACK	Reserved		
0xCF0 & 0xDF0	L4IACK	Reserved		
0xCF4 & 0xDF4	L5IACK	Reserved		
0xCF8 & 0xDF8	L6IACK	Reserved		
0xCFC & 0xDFC	L7IACK	Reserved		

## 13.2.1 Register Descriptions

### 13.2.1.1 Interrupt Pending Registers (IPRH $n$ , IPRL $n$ )

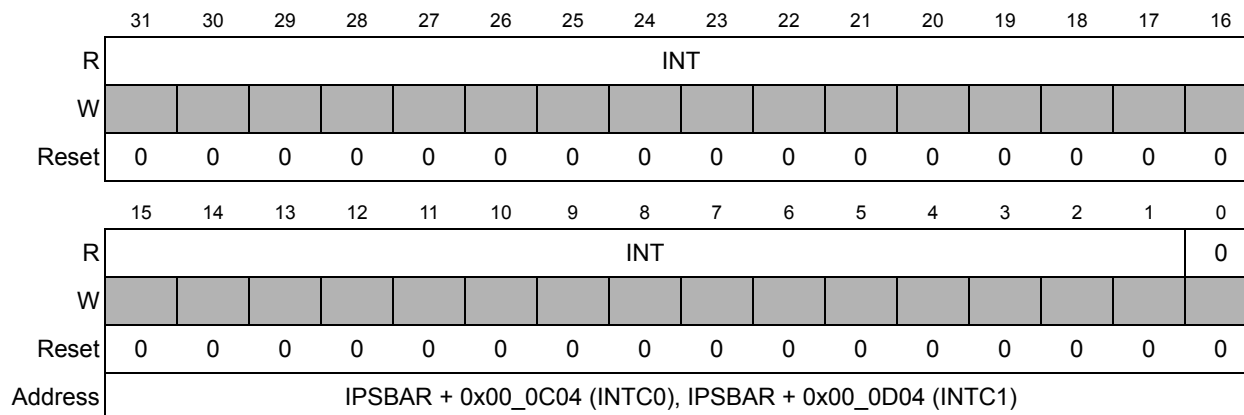
The IPRH $n$  and IPRL $n$  registers, [Figure 13-1](#) and [Figure 13-2](#), are each 32 bits in size, and provide a bit map for each interrupt request to indicate if there is an active request (1 = active request, 0 = no request) for the given source. The state of the interrupt mask register does not affect the IPR $n$ . The IPR $n$  is cleared by reset. The IPR $n$  is a read-only register, so any attempted write to this register is ignored. Bit 0 is not implemented and reads as a zero.

**Figure 13-1. Interrupt Pending Register High (IPRH $n$ )**



**Table 13-4. IPRH<sub>n</sub> Field Descriptions**

Bits	Name	Description
31–0	INT	<p>Interrupt pending. Each bit corresponds to an interrupt source. The corresponding IMRH<sub>n</sub> bit determines whether an interrupt condition can generate an interrupt. At every system clock, the IPRH<sub>n</sub> samples the signal generated by the interrupting source. The corresponding IPRH<sub>n</sub> bit reflects the state of the interrupt signal even if the corresponding IMRH<sub>n</sub> bit is set.</p> <p>0 The corresponding interrupt source does not have an interrupt pending  1 The corresponding interrupt source has an interrupt pending</p>

**Figure 13-2. Interrupt Pending Register Low (IPRL<sub>n</sub>)****Table 13-5. IPRL<sub>n</sub> Field Descriptions**

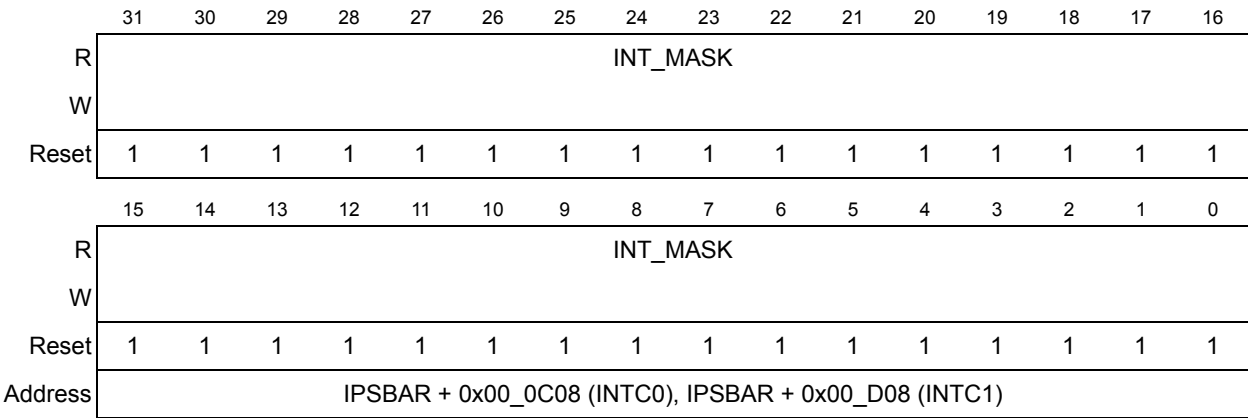
Bits	Name	Description
31–1	INT	<p>Interrupt Pending. Each bit corresponds to an interrupt source. The corresponding IMRL<sub>n</sub> bit determines whether an interrupt condition can generate an interrupt. At every system clock, the IPRL<sub>n</sub> samples the signal generated by the interrupting source. The corresponding IPRL<sub>n</sub> bit reflects the state of the interrupt signal even if the corresponding IMRL<sub>n</sub> bit is set.</p> <p>0 The corresponding interrupt source does not have an interrupt pending  1 The corresponding interrupt source has an interrupt pending</p>
0	—	Reserved, should be cleared.

### 13.2.1.2 Interrupt Mask Register (IMRH<sub>n</sub>, IMRL<sub>n</sub>)

The IMRH<sub>n</sub> and IMRL<sub>n</sub> registers are each 32 bits in size and provide a bit map for each interrupt to allow the request to be disabled (1 = disable the request, 0 = enable the request). The IMR<sub>n</sub> is set to all ones by reset, disabling all interrupt requests. The IMR<sub>n</sub> can be read and written. A write that sets bit 0 of the IMR forces the other 63 bits to be set, disabling all interrupt sources, and providing a global mask-all capability.

### NOTE

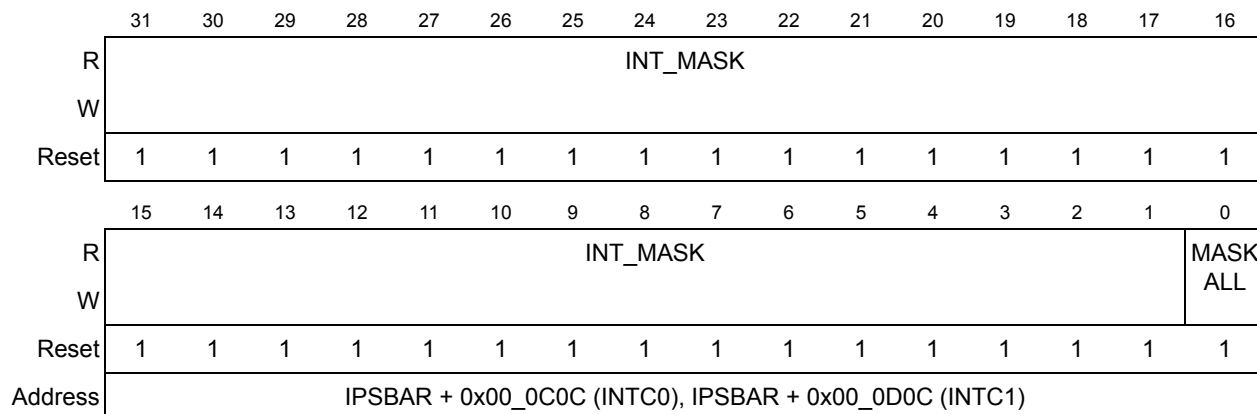
If an interrupt source is being masked in the interrupt controller mask register (IMR) or a module's interrupt mask register while the interrupt mask in the status register (SR[I]) is set to a value lower than the interrupt's level, a spurious interrupt may occur. This is because by the time the status register acknowledges this interrupt, the interrupt has been masked. A spurious interrupt is generated because the CPU cannot determine the interrupt source. To avoid this situation for interrupts sources with levels 1-6, first write a higher level interrupt mask to the status register, before setting the mask in the IMR or the module's interrupt mask register. After the mask is set, return the interrupt mask in the status register to its previous value. Since level 7 interrupts cannot be disabled in the status register prior to masking, use of the IMR or module interrupt mask registers to disable level 7 interrupts is not recommended.



**Figure 13-3. Interrupt Mask Register High (IMRH<sub>n</sub>)**

**Table 13-6. IMRH<sub>n</sub> Field Descriptions**

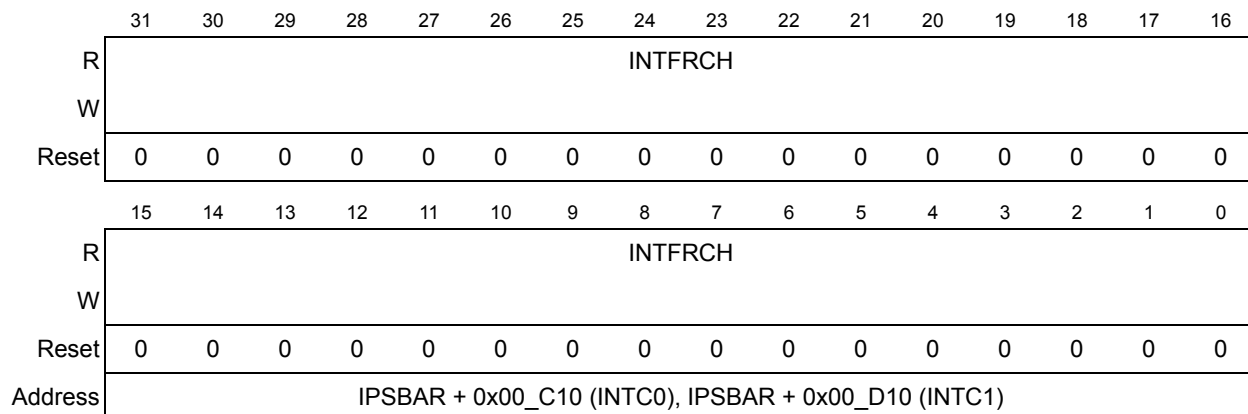
Bits	Name	Description
31–0	INT_MASK	<p>Interrupt mask. Each bit corresponds to an interrupt source. The corresponding IMRH<sub>n</sub> bit determines whether an interrupt condition can generate an interrupt. The corresponding IPRH<sub>n</sub> bit reflects the state of the interrupt signal even if the corresponding IMRH<sub>n</sub> bit is set.</p> <p>0 The corresponding interrupt source is not masked</p> <p>1 The corresponding interrupt source is masked</p>

**Figure 13-4. Interrupt Mask Register Low (IMRL<sub>n</sub>)****Table 13-7. IMRL<sub>n</sub> Field Descriptions**

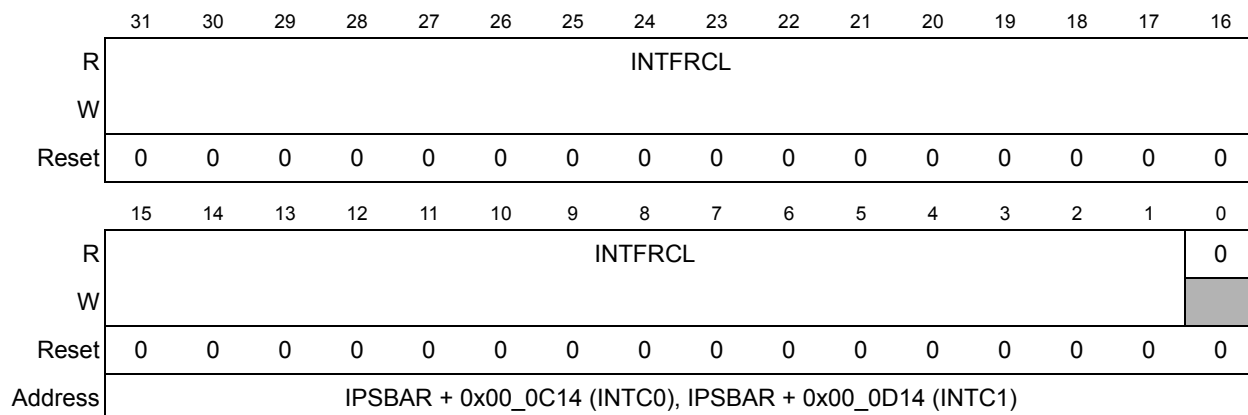
Bits	Name	Description
31–1	INT_MASK	Interrupt mask. Each bit corresponds to an interrupt source. The corresponding IMRL <sub>n</sub> bit determines whether an interrupt condition can generate an interrupt. The corresponding IPRL <sub>n</sub> bit reflects the state of the interrupt signal even if the corresponding IMRL <sub>n</sub> bit is set. 0 The corresponding interrupt source is not masked 1 The corresponding interrupt source is masked
0	MASKALL	Mask all interrupts. Setting this bit will force the other 63 bits of the IMRH <sub>n</sub> and IMRL <sub>n</sub> to ones, disabling all interrupt sources, and providing a global mask-all capability.

### 13.2.1.3 Interrupt Force Registers (INTFRCH<sub>n</sub>, INTFRCL<sub>n</sub>)

The INTFRCH<sub>n</sub> and INTFRCL<sub>n</sub> registers are each 32 bits in size and provide a mechanism to allow software generation of interrupts for each possible source for functional or debug purposes. The system design may reserve one or more sources to allow software to self-schedule interrupts by forcing one or more of these bits (1 = force request, 0 = negate request) in the appropriate INTFRC<sub>n</sub> register. The assertion of an interrupt request via the INTFRC<sub>n</sub> register is not affected by the interrupt mask register. The INTFRC<sub>n</sub> register is cleared by reset.

**Figure 13-5. Interrupt Force Register High (INTFRCH<sub>n</sub>)****Table 13-8. INTFRCH<sub>n</sub> Field Descriptions**

Bits	Name	Description
31–0	INTFRCH	Interrupt force. Allows software generation of interrupts for each possible source for functional or debug purposes. 0 No interrupt forced on corresponding interrupt source 1 Force an interrupt on the corresponding source

**Figure 13-6. Interrupt Force Register High (INTFRCH<sub>n</sub>)****Table 13-9. INTFRCL<sub>n</sub> Field Descriptions**

Bits	Name	Description
31–1	INTFRCL	Interrupt force. Allows software generation of interrupts for each possible source for functional or debug purposes. 0 No interrupt forced on corresponding interrupt source 1 Force an interrupt on the corresponding source
0	—	Reserved, should be cleared.

### 13.2.1.4 Interrupt Request Level Register (IRLR<sub>n</sub>)

This 7-bit register is updated each machine cycle and represents the current interrupt requests for each interrupt level, where bit 7 corresponds to level 7, bit 6 to level 6, etc. This register output from both interrupt controllers (INTC0 & INTC1) are combined encoded into the 3-bit priority interrupt level driven to the processor core.

	7	6	5	4	3	2	1	0
R	IRQ							0
W								
Reset	0	0	0	0	0	0	0	0
Address	IPSBAR + 0x00_0C18 (INTC0); IPSBAR + 0x00_0D18 (INTC1)							

**Figure 13-7. Interrupt Request Level Register (IRLR<sub>n</sub>)**

**Table 13-10. IRQLR<sub>n</sub> Field Descriptions**

Bits	Name	Description
7–1	IRQ	Interrupt requests. Represents the prioritized active interrupts for each level. 0 There are no active interrupts at this level 1 There is an active interrupt at this level
0	—	Reserved

### 13.2.1.5 Interrupt Acknowledge Level and Priority Register (IACKLPR<sub>n</sub>)

Each time an IACK is performed, the interrupt controller responds with the vector number of the highest priority source within the level being acknowledged. In addition to providing the vector number directly for the byte-sized IACK read, this 8-bit register is also loaded with information about the interrupt level and priority being acknowledged. This register provides the association between the acknowledged “physical” interrupt request number and the programmed interrupt level/priority. The contents of this read-only register are described in [Figure 13-8](#) and [Table 13-11](#).

	7	6	5	4	3	2	1	0
R	0	LEVEL			PRI			
W								
Reset	0	0	0	0	0	0	0	0
Address	IPSBAR + 0x00_0C19 (INTC0); IPSBAR + 0x00_0D19 (INTC1)							

**Figure 13-8. IACK Level and Priority Register (IACKLPR<sub>n</sub>)**

**Table 13-11. IACKLPR<sub>n</sub> Field Descriptions**

Bits	Name	Description
7	—	Reserved
6–4	LEVEL	Interrupt level. Represents the interrupt level currently being acknowledged.
3–0	PRI	Interrupt Priority. Represents the priority within the interrupt level of the interrupt currently being acknowledged. 0 Priority 0 1 Priority 1 2 Priority 2 3 Priority 3 4 Priority 4 5 Priority 5 6 Priority 6 7 Priority 7 8 Mid-Point Priority associated with the fixed level interrupts only

### 13.2.1.6 Interrupt Control Register (ICR<sub>*n*x</sub>, (*x* = 1, 2,..., 63))

Each ICR<sub>*n*x</sub> specifies the interrupt level (1–7) and the priority within the level (0–7). All ICR<sub>*n*x</sub> registers can be read, but only ICR<sub>*n*8</sub> to ICR<sub>*n*63</sub> can be written. It is the responsibility of the software to program the ICR<sub>*n*x</sub> registers with unique and non-overlapping level and priority definitions. Failure to program the ICR<sub>*n*x</sub> registers in this manner can result in undefined behavior. If a specific interrupt request is completely unused, the ICR<sub>*n*x</sub> value can remain in its reset (and disabled) state.

	7	6	5	4	3	2	1	0
R:	0	0	IL			IP		
W			See Note					
Reset	0	0	0	0	0	0	0	0
Address	See <a href="#">Table 13-2</a> and <a href="#">Table 13-3</a> for register offsets							

**Note:** Read only for ICR<sub>*n*1</sub>-ICR<sub>*n*7</sub>, else Read/Write

**Note:** It is the responsibility of the software to program the ICR<sub>*n*x</sub> registers with unique and non-overlapping level and priority definitions. Failure to program the ICR<sub>*n*x</sub> registers in this manner can result in undefined behavior. If a specific interrupt request is completely unused, the ICR<sub>*n*x</sub> value can remain in its reset (and disabled) state.

**Figure 13-9. Interrupt Control Register (ICR<sub>*n*x</sub>)****Table 13-12. ICR<sub>*n*x</sub> Field Descriptions**

Bits	Name	Description
7–6	—	Reserved, should be cleared.

**Table 13-12. ICR<sub>n</sub>x Field Descriptions (Continued)**

Bits	Name	Description
5–3	IL	Interrupt level. Indicates the interrupt level assigned to each interrupt input.
2–0	IP	Interrupt priority. Indicates the interrupt priority for internal modules within the interrupt-level assignment. 000 represents the lowest priority and 111 represents the highest. For the fixed level interrupt sources, the priority is fixed at the midpoint for the level, and the IP field will always read as 000.

### 13.2.1.6.1 Interrupt Sources

The following tables list the interrupt sources for each interrupt request line for INTC0 and INTC1.

**Table 13-13. Interrupt Source Assignment For INTC0**

Source	Module	Flag	Source Description	Flag Clearing Mechanism
1	EPORT	EPF1	Edge port flag 1	Write EPF1 = 1
2		EPF2	Edge port flag 2	Write EPF2 = 1
3		EPF3	Edge port flag 3	Write EPF3 = 1
4		EPF4	Edge port flag 4	Write EPF4 = 1
5		EPF5	Edge port flag 5	Write EPF5 = 1
6		EPF6	Edge port flag 6	Write EPF6 = 1
7		EPF7	Edge port flag 7	Write EPF7 = 1
8	SCM	SWT1	Software watchdog timeout	Cleared when service complete.
9	DMA	DONE	DMA Channel 0 transfer complete	Write DONE = 1
10		DONE	DMA Channel 1 transfer complete	Write DONE = 1
11		DONE	DMA Channel 2 transfer complete	Write DONE = 1
12		DONE	DMA Channel 3 transfer complete	Write DONE = 1
13	UART0	INT	UART0 interrupt	Automatically cleared
14	UART1	INT	UART1 interrupt	Automatically cleared
15	UART2	INT	UART2 interrupt	Automatically cleared
16	Not used			
17	I <sup>2</sup> C	IIF	I <sup>2</sup> C interrupt	Write IIF = 0
18	QSPI	INT	QSPI interrupt	Write 1 to appropriate QIR bit
19	TMR0	INT	TMR0 interrupt	Write 1 to appropriate TER0 bit
20	TMR1	INT	TMR1 interrupt	Write 1 to appropriate TER1 bit
21	TMR2	INT	TMR2 interrupt	Write 1 to appropriate TER2 bit
22	TMR3	INT	TMR3 interrupt	Write 1 to appropriate TER3 bit

**Table 13-13. Interrupt Source Assignment For INTC0 (Continued)**

Source	Module	Flag	Source Description	Flag Clearing Mechanism
23	FEC0	X_INTF	Transmit frame interrupt	Write X_INTF = 1
24		X_INTB	Transmit buffer interrupt	Write X_INTB = 1
25		UN	Transmit FIFO underrun	Write UN = 1
26		RL	Collision retry limit	Write RL = 1
27		R_INTF	Receive frame interrupt	Write R_INTF = 1
28		R_INTB	Receive buffer interrupt	Write R_INTB = 1
29		MII	MII interrupt	Write MII = 1
30		LC	Late collision	Write LC = 1
31		HBERR	Heartbeat error	Write HBERR = 1
32		GRA	Graceful stop complete	Write GRA = 1
33		EBERR	Ethernet bus error	Write EBERR = 1
34		BABT	Babbling transmit error	Write BABT = 1
35		BABR	Babbling receive error	Write BABR = 1
36	PIT0	PIF	PIT Interrupt Flag	Write PIF = 1 or write PMR
37	PIT1	PIF	PIT Interrupt Flag	Write PIF = 1 or write PMR
38	PIT2	PIF	PIT Interrupt Flag	Write PIF = 1 or write PMR
39	PIT3	PIF	PIT Interrupt Flag	Write PIF = 1 or write PMR
40	RNG	EI	RNG interrupt flag	Write RNGCR[CI] = 1
41	SKHA	INT	SKHA interrupt flag	Write SKCMR[CI] = 1
42	MDHA	MI	MDHA interrupt flag	Write MDCMR[CI] = 1
43	USB	USB	USB Interrupt	writing a '1' to the appropriate bit in USB_INTR
44		EP0	Endpoint 0	writing a '1' to the appropriate bit in USB_EP0_INTR
45		EP1	Endpoint 1	writing a '1' to the appropriate bit in USB_EP1_INTR
46		EP2	Endpoint 2	writing a '1' to the appropriate bit in USB_EP2_INTR
47		EP3	Endpoint 3	writing a '1' to the appropriate bit in USB_EP3_INTR
48–63	Not used			

**Table 13-14. Interrupt Source Assignment for INTC1**

Source	Module	Flag	Source Description	Flag Clearing Mechanism
1–22	Not used			



**Table 13-14. Interrupt Source Assignment for INTC1 (Continued)**

Source	Module	Flag	Source Description	Flag Clearing Mechanism
23	FEC1	X_INTF	Transmit frame interrupt	Write X_INTF = 1
24		X_INTB	Transmit buffer interrupt	Write X_INTB = 1
25		UN	Transmit FIFO underrun	Write UN = 1
26		RL	Collision retry limit	Write RL = 1
27		R_INTF	Receive frame interrupt	Write R_INTF = 1
28		R_INTB	Receive buffer interrupt	Write R_INTB = 1
29		MII	MII interrupt	Write MII = 1
30		LC	Late collision	Write LC = 1
31		HBERR	Heartbeat error	Write HBERR = 1
32		GRA	Graceful stop complete	Write GRA = 1
33		EBERR	Ethernet bus error	Write EBERR = 1
34		BABT	Babbling transmit error	Write BABT = 1
35		BABR	Babbling receive error	Write BABR = 1
36–63	Not used			

### 13.2.1.7 Software and Level *n* IACK Registers (SWIACKR, L1IACK–L7IACK)

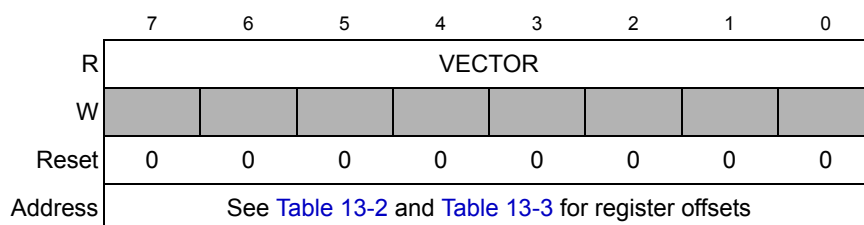
The eight IACK registers can be explicitly addressed via the CPU, or implicitly addressed via a processor-generated interrupt acknowledge cycle during exception processing. In either case, the interrupt controller's actions are very similar.

First, consider an IACK cycle to a specific level: that is, a level-*n* IACK. When this type of IACK arrives in the interrupt controller, the controller examines all the currently-active level *n* interrupt requests, determines the highest priority within the level, and then responds with the unique vector number corresponding to that specific interrupt source. The vector number is supplied as the data for the byte-sized IACK read cycle. In addition to providing the vector number, the interrupt controller also loads the level and priority number for the level into the IACKLPR register, where it may be retrieved later.

This interrupt controller design also supports the concept of a software IACK. A software IACK is a useful concept that allows an interrupt service routine to determine if there are other pending interrupts so that the overhead associated with interrupt exception processing (including machine state save/restore functions) can be minimized. In general, the software IACK is performed near the end of an interrupt service routine, and if there are additional active interrupt sources, the current interrupt service routine (ISR) passes control to the appropriate service routine, but without taking another interrupt exception.

When the interrupt controller receives a software IACK read, it returns the vector number associated with the highest level, highest priority unmasked interrupt source for that interrupt controller. The IACKLPR register is also loaded as the software IACK is performed. If there are no active sources, the interrupt controller returns an all-zero vector as the operand. For this situation, the IACKLPR register is also cleared.

In addition to the software IACK registers within each interrupt controller, there are global software IACK registers. A read from the global SWIACK will return the vector number for the highest level and priority unmasked interrupt source from all interrupt controllers. A read from one of the  $L_n$ IACK registers will return the vector for the highest priority unmasked interrupt within a level for all interrupt controllers.



**Figure 13-10. Software and Level  $n$  IACK Registers (SWIACKR, L1IACK–L7IACK)**

**Table 13-15. SWIACK and L1IACK-L7IACK Field Descriptions**

Bits	Name	Description
7–0	VECTOR	Vector number. A read from the SWIACK register returns the vector number associated with the highest level, highest priority unmasked interrupt source. A read from one of the $L_n$ IACK registers returns the highest priority unmasked interrupt source within the level.

## 13.3 Prioritization Between Interrupt Controllers

The interrupt controllers have a fixed priority, where INTC0 has the highest priority, and INTC1 has the lowest priority. If both interrupt controllers have active interrupts at the same level and priority, then the INTC0 interrupt will be serviced first. If INTC1 has an active interrupt that has a higher level or priority than the highest INTC0 interrupt, then the INTC1 interrupt will be serviced first.

## 13.4 Low-Power Wakeup Operation

The System Control Module (SCM) contains an 8-bit low-power interrupt control register (LPICR) used explicitly for controlling the low-power stop mode. This register must explicitly be programmed by software to enter low-power mode.

Each interrupt controller provides a special combinatorial logic path to provide a special wake-up signal to exit from the low-power stop mode. This special mode of operation works as follows:

- First, LPICR[XLPM\_IPL] is loaded with the mask level that will be specified while the core is in stop mode. LPICR[ENBSTOP] must be set to enable this mode of operation.

#### NOTE

The wakeup mask level taken from LPICR[XLPM\_IPL] is adjusted by hardware to allow a level 7 IRQ to generate a wakeup. That is, the wakeup mask value used by the interrupt controller must be in the range of 0–6.

- Second, the processor executes a STOP instruction which places it in stop mode. Once the processor is stopped, each interrupt controller enables a special logic path which evaluates the incoming interrupt sources in a purely combinatorial path; that is, there are no clocked storage elements. If an active interrupt request is asserted and the resulting interrupt level is greater than the mask value contained in LPICR[XLPM\_IPL], then each interrupt controller asserts the wake-up output signal, which is routed to the SCM where it is combined with the wakeup signals from the other interrupt controller and then to the PLL module to re-enable the device's clock trees and resume processing.

For more information, see [Section 8.2.1.1, “Low-Power Interrupt Control Register \(LPICR\).”](#)



# Chapter 14

## Edge Port Module (EPORT)

### 14.1 Introduction

The edge port module (EPORT) has seven external interrupt pins,  $\overline{\text{IRQ}}7\text{--}\overline{\text{IRQ}}1$ . Each pin can be configured individually as a level-sensitive interrupt pin, an edge-detecting interrupt pin (rising edge, falling edge, or both), or a general-purpose input/output (I/O) pin. See [Figure 14-1](#).

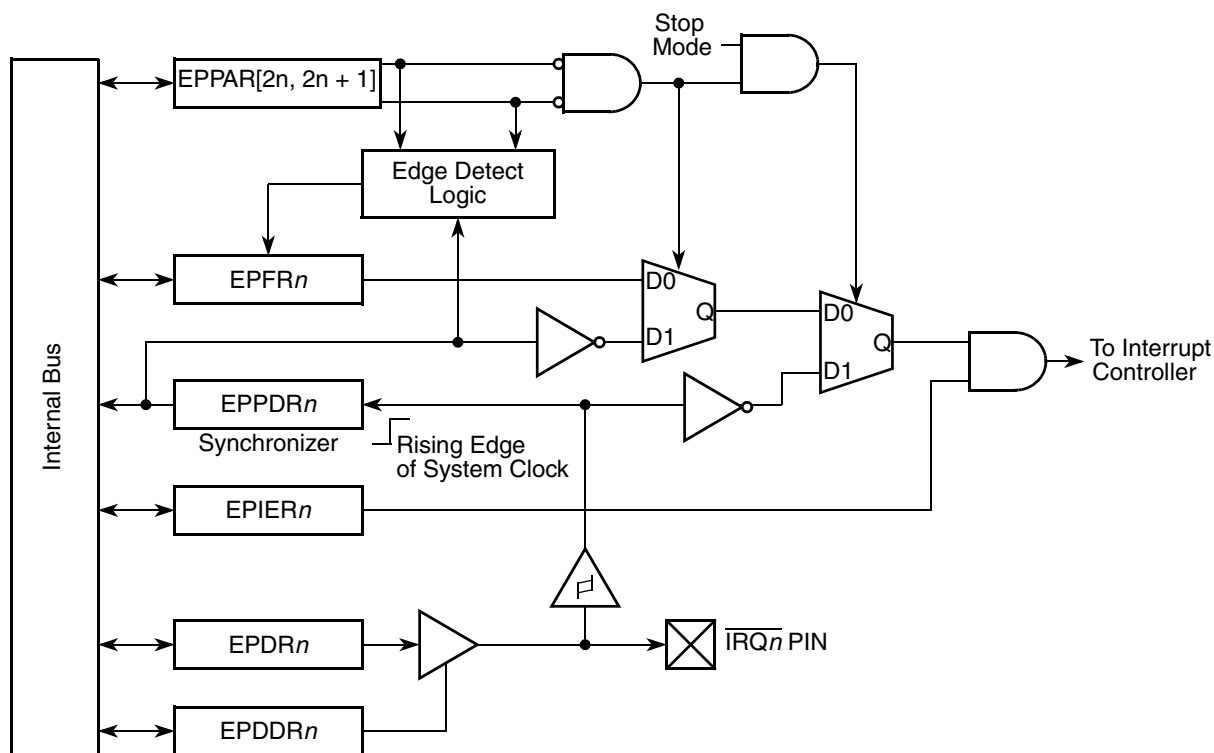


Figure 14-1. EPORT Block Diagram

### 14.2 Low-Power Mode Operation

This section describes the operation of the EPORT module in low-power modes. For more information on low-power modes, see [Chapter 8, “Power Management.”](#) [Table 14-1](#) shows EPORT module operation in low-power modes, and describes how this module may exit from each mode.

**NOTE**

The low-power interrupt control register (LPICR) in the System Control Module specifies the interrupt level at or above which is needed to bring the device out of a low-power mode.

**Table 14-1. Edge Port Module Operation in Low-power Modes**

Low-power Mode	EPORT Operation	Mode Exit
Wait	Normal	Any $\overline{IRQn}$ Interrupt at or above level in LPICR
Doze	Normal	Any $\overline{IRQn}$ Interrupt at or above level in LPICR
Stop	Level-sensing Only	Any $\overline{IRQn}$ Interrupt set for level-sensing at or above level in LPICR

In wait and doze modes, the EPORT module continues to operate as it does in run mode. It may be configured to exit the low-power modes by generating an interrupt request on either a selected edge or a low level on an external pin. In stop mode, there are no clocks available to perform the edge-detect function. Only the level-detect logic is active (if configured) to allow any low level on the external interrupt pin to generate an interrupt (if enabled) to exit stop mode.

**NOTE**

The input pin synchronizer is bypassed for the level-detect logic since no clocks are available.

### 14.3 Interrupt/General-Purpose I/O Pin Descriptions

All pins default to general-purpose input pins at reset. The pin value is synchronized to the rising edge of CLKOUT when read from the EPORT pin data register (EPPDR). The values used in the edge/level detect logic are also synchronized to the rising edge of CLKOUT. These pins use Schmitt triggered input buffers which have built in hysteresis designed to decrease the probability of generating false edge-triggered interrupts for slow rising and falling input signals.

When a pin is configured as an output, it is driven to a state whose level is determined by the corresponding bit in the EPORT data register (EPDR). All bits in the EPDR are high at reset.

### 14.4 Memory Map/Register Definition

This subsection describes the memory map and register structure. Refer to [Table 14-2](#) for a description of the EPORT memory map. The EPORT has an IPSBAR offset for base address of 0x0013\_0000.

**Table 14-2. Edge Port Module Memory Map**

IPSBAR Offset	Bits 15–8	Bits 7–0	Access <sup>1</sup>
0x13_0000	EPORT Pin Assignment Register (EPPAR)		S
0x13_0002	EPORT Data Direction Register (EPDDR)	EPORT Interrupt Enable Register (EPIER)	S
0x13_0004	EPORT Data Register (EPDR)	EPORT Pin Data Register (EPPDR)	S/U
0x13_0006	EPORT Flag Register (EPFR)	Reserved <sup>2</sup>	S/U

<sup>1</sup> S = CPU supervisor mode access only. S/U = CPU supervisor or user mode access. User mode accesses to supervisor only addresses have no effect and result in a cycle termination transfer error.

<sup>2</sup> Writing to reserved address locations has no effect, and reading returns 0s.

## 14.4.1 Register Description

The EPORT programming model consists of these registers:

- The EPORT pin assignment register (EPPAR) controls the function of each pin individually.
- The EPORT data direction register (EPDDR) controls the direction of each one of the pins individually.
- The EPORT interrupt enable register (EPIER) enables interrupt requests for each pin individually.
- The EPORT data register (EPDR) holds the data to be driven to the pins.
- The EPORT pin data register (EPPDR) reflects the current state of the pins.
- The EPORT flag register (EPFR) individually latches EPORT edge events.

### 14.4.1.1 EPORT Pin Assignment Register (EPPAR)

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	EPPA7		EPPA6		EPPA5		EPPA4		EPPA3		EPPA2		EPPA1		0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Address	IPSBAR + 0x13_0000															

**Figure 14-2. EPORT Pin Assignment Register (EPPAR)**

**Table 14-3. EPPAR Field Descriptions**

Bits	Name	Description
15–2	EPPAn	<p>EPORT pin assignment select fields. The read/write EPPAn fields configure EPORT pins for level detection and rising and/or falling edge detection.</p> <p>Pins configured as level-sensitive are inverted so that a logic 0 on the external pin represents a valid interrupt request. Level-sensitive interrupt inputs are not latched. To guarantee that a level-sensitive interrupt request is acknowledged, the interrupt source must keep the signal asserted until acknowledged by software. Level sensitivity must be selected to bring the device out of stop mode with an IRQn interrupt.</p> <p>Pins configured as edge-triggered are latched and need not remain asserted for interrupt generation. A pin configured for edge detection can trigger an interrupt regardless of its configuration as input or output.</p> <p>Interrupt requests generated in the EPORT module can be masked by the interrupt controller module. EPPAR functionality is independent of the selected pin direction.</p> <p>Reset clears the EPPAn fields.</p> <p>00 Pin IRQn level-sensitive  01 Pin IRQn rising edge triggered  10 Pin IRQn falling edge triggered  11 Pin IRQn both falling edge and rising edge triggered</p>
1–0	—	Reserved, should be cleared.

#### 14.4.1.2 EPORT Data Direction Register (EPDDR)

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	EPDD7		EPDD6		EPDD5		EPDD4		EPDD3		EPDD2		EPDD1		0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Address	IPSBAR + 0x13_0002															

**Figure 14-3. EPORT Data Direction Register (EPDDR)****Table 14-4. EPDD Field Descriptions**

Bits	Name	Description
7–1	EPDDn	<p>Setting any bit in the EPDDR configures the corresponding pin as an output. Clearing any bit in EPDDR configures the corresponding pin as an input. Pin direction is independent of the level/edge detection configuration. Reset clears EPDD7–EPDD1.</p> <p>To use an EPORT pin as an external interrupt request source, its corresponding bit in EPDDR must be clear. Software can generate interrupt requests by programming the EPORT data register when the EPDDR selects output.</p> <p>0 Corresponding EPORT pin configured as input  1 Corresponding EPORT pin configured as output</p>
0	—	Reserved, should be cleared.



### 14.4.1.3 Edge Port Interrupt Enable Register (EPIER)

	7	6	5	4	3	2	1	0
R	EPIE7	EPIE6	EPIE5	EPIE4	EPIE3	EPIE2	EPIE1	0
W								
Reset	0	0	0	0	0	0	0	0
Address	IPSBAR + 0x13_0003							

**Figure 14-4. EPORT Port Interrupt Enable Register (EPIER)**

**Table 14-5. EPIER Field Descriptions**

Bits	Name	Description
7–1	EPIE $n$	Edge port interrupt enable bits enable EPORT interrupt requests. If a bit in EPIER is set, EPORT generates an interrupt request when: <ul style="list-style-type: none"> <li>The corresponding bit in the EPORT flag register (EPFR) is set or later becomes set.</li> <li>The corresponding pin level is low and the pin is configured for level-sensitive operation.</li> </ul> Clearing a bit in EPIER negates any interrupt request from the corresponding EPORT pin. Reset clears EPIE7–EPIE1. <ul style="list-style-type: none"> <li>0 Interrupt requests from corresponding EPORT pin disabled</li> <li>1 Interrupt requests from corresponding EPORT pin enabled</li> </ul>
0	—	Reserved, should be cleared.

### 14.4.1.4 Edge Port Data Register (EPDR)

	7	6	5	4	3	2	1	0
R	EPD7	EPD6	EPD5	EPD4	EPD3	EPD2	EPD1	0
W								
Reset	1	1	1	1	1	1	1	1
Address	IPSBAR + 0x13_0004							

**Figure 14-5. EPORT Port Data Register (EPDR)**

**Table 14-6. EPDR Field Descriptions**

Bits	Name	Description
7–1	EPD $n$	Edge port data bits. Data written to EPDR is stored in an internal register; if any pin of the port is configured as an output, the bit stored for that pin is driven onto the pin. Reading EPDR returns the data stored in the register. Reset sets EPD7–EPD1.
0	—	Reserved, should be cleared.

### 14.4.1.5 Edge Port Pin Data Register (EPPDR)

	7	6	5	4	3	2	1	0
R	EPPD7	EPPD6	EPPD5	EPPD4	EPPD3	EPPD2	EPPD1	0
W								
Reset	Current pin state							0
Address	IPSBAR + 0x13_0005							

**Figure 14-6. EPORT Port Pin Data Register (EPPDR)**

**Table 14-7. EPPDR Field Descriptions**

Bits	Name	Description
7–1	EPPD $n$	Edge port pin data bits. The read-only EPPDR reflects the current state of the EPORT pins IRQ7–IRQ1. Writing to EPPDR has no effect, and the write cycle terminates normally. Reset does not affect EPPDR.
0	—	Reserved, should be cleared.

### 14.4.1.6 Edge Port Flag Register (EPFR)

	7	6	5	4	3	2	1	0
R	EPF7	EPF6	EPF5	EPF4	EPF3	EPF2	EPF1	0
W								
Reset	0	0	0	0	0	0	0	0
Address	IPSBAR + 0x13_0006							

**Figure 14-7. EPORT Port Flag Register (EPFR)**

**Table 14-8. EPFR Field Descriptions**

Bits	Name	Description
7–1	EPF $n$	Edge port flag bits. When an EPORT pin is configured for edge triggering, its corresponding read/write bit in EPFR indicates that the selected edge has been detected. Reset clears EPF7–EPF1. Bits in this register are set when the selected edge is detected on the corresponding pin. A bit remains set until cleared by writing a 1 to it. Writing 0 has no effect. If a pin is configured as level-sensitive (EPPAR $n$ = 00), pin transitions do not affect this register. 0 Selected edge for $\overline{\text{IRQ}}n$ pin has not been detected. 1 Selected edge for IRQ $n$ pin has been detected.
0	—	Reserved, should be cleared.

# Chapter 15

## Chip Select Module

### 15.1 Introduction

This chapter describes the chip select module, including the operation and programming model of the chip select registers, which include the chip select address, mask, and control registers.

#### NOTE

Unless otherwise noted, in this chapter, “clock” refers to the CLKOUT used for the bus ( $f_{\text{sys}}/2$ ).

#### 15.1.1 Overview

The following list summarizes the key chip select features:

- Eight independent, user-programmable chip select signals ( $\overline{\text{CS}}[7:0]$ ) that can interface with external SRAM, PROM, EPROM, EEPROM, Flash, and peripherals
- Address masking for 64-Kbyte to 4-Gbyte memory block sizes
- Enhanced secondary wait states
- Access error on writes to write-protect (WP) region

### 15.2 External Signal Description

This section describes the signals used by the chip select module.

#### 15.2.1 Chip Selects ( $\overline{\text{CS}}[7:0]$ )

Each CS<sub>n</sub> can be independently programmed for an address location as well as for masking, port size, read/write burst capability, wait-state generation, and internal/external termination. Only CS<sub>0</sub> is initialized at reset and may act as an external boot chip select to allow boot ROM to be at an external address space. Port size for  $\overline{\text{CS}}_0$  is configured by the logic levels of D[20:19] when  $\overline{\text{RSTOUT}}$  negates and  $\overline{\text{RCON}}$  is asserted.

#### 15.2.2 Output Enable ( $\overline{\text{OE}}$ )

Interfaces to memory or to peripheral devices and enables a read transfer. It is asserted and negated on the falling edge of the clock.  $\overline{\text{OE}}$  is asserted only when one of the chip selects matches for the current address decode.

### 15.2.3 Byte Strobes ( $\overline{BS}[3:2]$ )

These signals are individually programmed through the byte-enable mode bit,  $CSCR_n[BEM]$ , described in [Section 15.4.1.3, “Chip Select Control Registers \(CSCR0–CSCR7\)”](#).

These generated signals provide byte data select signals, which are decoded from the transfer size, A1, and A0 signals in addition to the programmed port size and burstability of the memory accessed, as shown below.

The below table also shows the interaction of the byte-enable/byte-write enables with related signals.

**Table 15-1. Byte Enables/Byte Write Enable Signal Settings**

Transfer Size	Port Size	A1	A0	$\overline{BS3}$	$\overline{BS2}$
				D[31:24]	D[23:16]
Byte	8-bit	0	0	0	1
		0	1	0	1
		1	0	0	1
		1	1	0	1
	16-bit	0	0	0	1
		0	1	1	0
		1	0	0	1
		1	1	1	0
Word	8-bit	0	0	0	1
		0	1	0	1
		1	0	0	1
		1	1	0	1
	16-bit	0	0	0	0
		1	0	0	0
Longword	8-bit	0	0	0	1
		0	1	0	1
		1	0	0	1
		1	1	0	1
	16-bit	0	0	0	0
		1	0	0	0
Line	8-bit	0	0	0	1
		0	1	0	1
		1	0	0	1
		1	1	0	1
	16-bit	0	0	0	0
		1	0	0	0

## 15.3 Chip Select Operation

Each chip select has a dedicated set of registers for configuration and control.

- Chip select address registers (CSAR<sub>n</sub>) control the base address of the chip select. See Section 15.4.1.1.
- Chip select mask registers (CSMR<sub>n</sub>) provide 16-bit address masking and access control. See Section 15.4.1.2.
- Chip select control registers (CSCR<sub>n</sub>) provide port size and burst capability indication, wait-state generation, and automatic acknowledge generation features. See Section 15.4.1.3.

$\overline{\text{CS0}}$  is a global chip select after reset and provides relocatable boot ROM capability.

### 15.3.1 General Chip Select Operation

When a bus cycle is initiated, the MCF5275 first compares its address with the base address and mask configurations programmed for chip selects 0–7 (configured in CSCR0–CSCR7) and DRAM blocks 0 and 1 (configured in DACR0 and DACR1). If the driven address matches a programmed chip select or DRAM block, the appropriate chip select is asserted or the DRAM block is selected using the specifications programmed in the respective configuration register. Otherwise, the following occurs:

- If the address and attributes do not match in CSAR or DACR, the MCF5275 runs an external burst-inhibited bus cycle with a default of external termination on a 16-bit port.
- Should an address and attribute match in multiple CSCRs, the matching chip select signals are driven; however, the chip select signals are driven during an external burst-inhibited bus cycle with external termination on a 16-bit port.
- If the address and attribute match both DACRs or a DACR and a CSAR, the operation is undefined.

Table 15-2 shows the type of access as a function of match in the CSARs and DACRs.

**Table 15-2. Accesses by Matches in CSARs and DACRs**

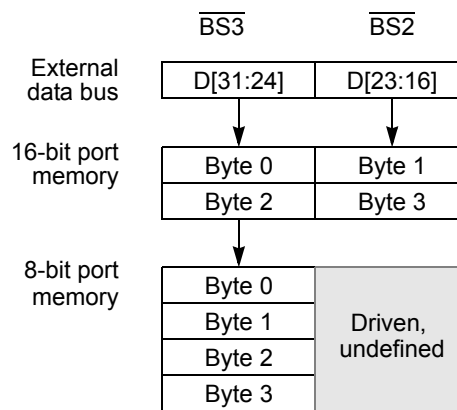
Number of CSCR Matches	Number of DACR Matches	Type of Access
0	0	External
1	0	Defined by CSAR
Multiple	0	External, burst-inhibited, 16-bit
0	1	Defined by DACRs
1	1	Undefined
Multiple	1	Undefined
0	Multiple	Undefined

**Table 15-2. Accesses by Matches in CSARs and DACRs (Continued)**

Number of CSCR Matches	Number of DACR Matches	Type of Access
1	Multiple	Undefined
Multiple	Multiple	Undefined

### 15.3.1.1 8- and 16-Bit Port Sizing

Static bus sizing is programmable through the port size bits, CSCR[PS]. See Section 15.4.1.3 for more information. Figure 15-1 shows the correspondence between the data bus and the external byte strobe control lines ( $\overline{BS}[3:2]$ ). Note that all byte lanes are driven, although the state of unused byte lanes is undefined.

**Figure 15-1. Connections for External Memory Port Sizes**

### 15.3.2 Enhanced Wait State Operation

The chip-select logic has been enhanced to add the notion of secondary wait-state counter values to be used after the initial wait-state value (where the existing wait state field becomes the initial access wait state) is applied to the first access. Two fields in the Chip-Select Control Registers (CSCR $_n$ ) are defined as the secondary read wait state (SRWS) value and the secondary write wait state (SWWS) value. The application of the secondary wait state values is only enabled when the auto-acknowledge feature is enabled, i.e., CSCR $_n$ [AA] = 1.

High-performance memory devices typically require a x-y-y-y response where x is the initial wait-state value and y is the secondary wait-state value. For non-SDRAM memory devices likely to be connected to a MCF5275 device, memory response times of {5-9}-{1-2}-{1-2}-{1-2} are typical. This function is supported with the secondary wait state counters.

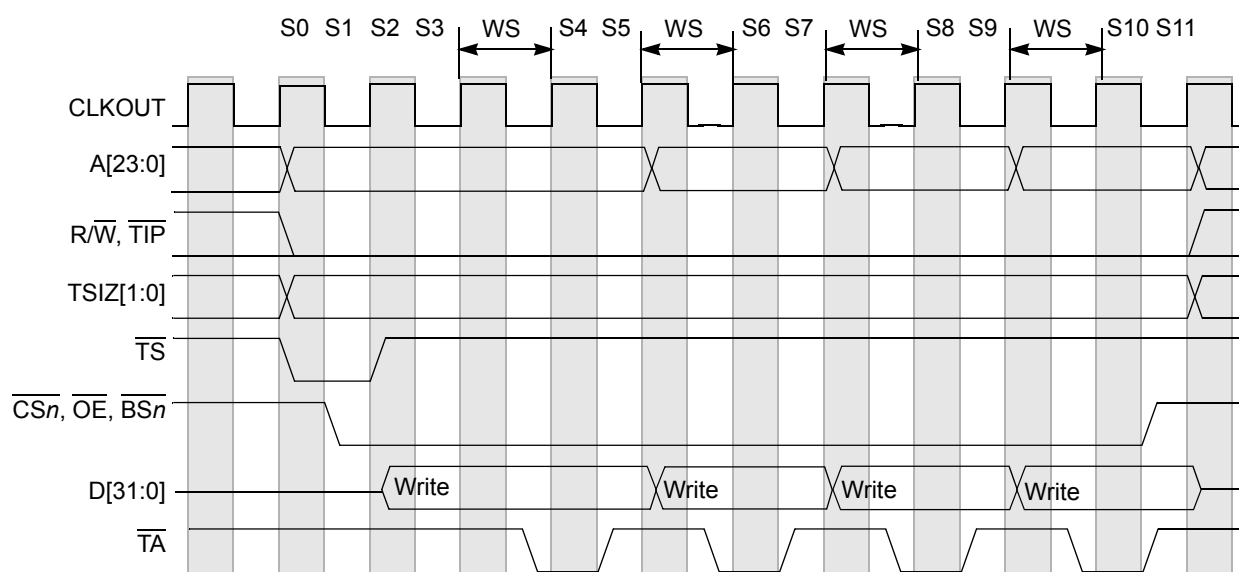


Figure 15-2. Secondary Wait State Writes

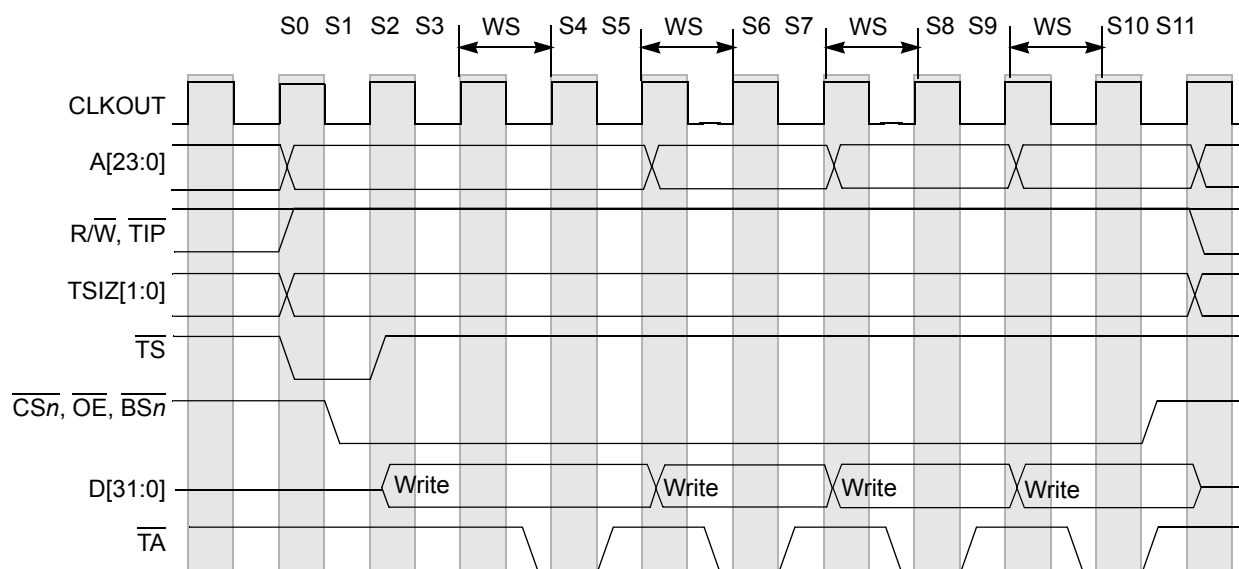


Figure 15-3. Secondary Wait State Reads

Each CSCR, shown in [Figure 15-6](#), controls the auto-acknowledge, port size, burst capability, and activation of each chip select.

### 15.3.2.1 External Boot Chip Select Operation

$\overline{CS0}$ , the external boot chip select, allows address decoding for boot ROM before system initialization. Its operation differs from other external chip select outputs after system reset.

After system reset,  $\overline{CS0}$  is asserted for every external access. No other chip select can be used until the valid bit, CSMR0[V], is set, at which point  $\overline{CS0}$  functions as configured and  $\overline{CS}[7:1]$  can be

used. At reset, the port size function of the external boot chip select is determined by the logic levels of the inputs on D[20:19]. [Table 15-3](#) and [Table 15-3](#) list the various reset encodings for the configuration signals multiplexed with D[20:19].

**Table 15-3. D[20:19] External Boot Chip Select Configuration**

D[20:19]	Boot Device/Data Port Size
00	
01	External (16-bit)
10	External (8-bit)
11	

Provided the required address range is in the chip select address register (CSAR0),  $\overline{CS0}$  can be programmed to continue decoding for a range of addresses after the CSMR0[V] is set, after which the external boot chip select can be restored only by a system reset.

## 15.4 Memory Map/Register Definition

[Table 15-4](#) shows the chip select register memory map. Reading reserved locations returns zeros.

**Table 15-4. Chip Select Registers**

IPSBAR Offset	[31:24]	[23:16]	[15:8]	[7:0]
0x00_0080	Chip select address register—bank 0 (CSAR0)		Reserved <sup>1</sup>	
0x00_0084	Chip select mask register—bank 0 (CSMR0)			
0x00_0088	Reserved <sup>1</sup>		Chip select control register—bank 0 (CSCR0)	
0x00_008C	Chip select address register—bank 1 (CSAR1)		Reserved <sup>1</sup>	
0x00_0090	Chip select mask register—bank 1 (CSMR1)			
0x00_0094	Reserved <sup>1</sup>		Chip select control register—bank 1 (CSCR1)	
0x00_0098	Chip select address register—bank 2 (CSAR2)		Reserved <sup>1</sup>	
0x00_009C	Chip select mask register—bank 2 (CSMR2)			
0x00_00A0	Reserved <sup>1</sup>		Chip select control register—bank 2 (CSCR2)	
0x00_00A4	Chip select address register—bank 3 (CSAR3)		Reserved <sup>1</sup>	
0x00_00A8	Chip select mask register—bank 3 (CSMR3)			
0x00_00A C	Reserved <sup>1</sup>		Chip select control register—bank 3 (CSCR3)	
0x00_00B0	Chip select address register—bank 4 (CSAR4)		Reserved <sup>1</sup>	
0x00_00B4	Chip select mask register—bank 4 (CSMR4)			
0x00_00B8	Reserved <sup>1</sup>		Chip select control register—bank 4 (CSCR4)	



**Table 15-4. Chip Select Registers (Continued)**

IPSBAR Offset	[31:24]	[23:16]	[15:8]	[7:0]
0x00_00B C	Chip select address register—bank 5 (CSAR5)		Reserved <sup>1</sup>	
0x00_00C0	Chip select mask register—bank 5 (CSMR5)			
0x00_00C4	Reserved <sup>1</sup>		Chip select control register—bank 5 (CSCR5)	
0x00_00C8	Chip select address register—bank 6 (CSAR6)		Reserved <sup>1</sup>	
0x00_00C C	Chip select mask register—bank 6 (CSMR6)			
0x00_00D0	Reserved <sup>1</sup>		Chip select control register—bank 6 (CSCR6)	
0x00_00D4	Chip-select address register—bank 7 (CSAR7)		Reserved <sup>1</sup>	
0x00_00D8	Chip-select mask register—bank 7 (CSMR7)			
0x00_00D C	Reserved <sup>1</sup>		Chip-select control register—bank 7 (CSCR7)	

<sup>1</sup> Addresses not assigned to a register and undefined register bits are reserved for expansion. Write accesses to these reserved address spaces and reserved register bits have no effect.

## 15.4.1 Chip Select Module Registers

The chip select module is programmed through the chip select address registers (CSAR0–CSAR7), chip select mask registers (CSMR0–CSMR7), and the chip select control registers (CSCR0–CSCR7).

### 15.4.1.1 Chip Select Address Registers (CSAR0–CSAR7)

The CSARs, [Figure 15-4](#), specify the chip select base addresses.

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	BA															
W																
Reset	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Address	IPSBAR + 0x00_0080 (CSAR0); IPSBAR + 0x00_008C (CSAR1); IPSBAR + 0x00_0098 (CSAR2); IPSBAR + 0x00_00A4 (CSAR3); IPSBAR + 0x00_00B0 (CSAR4); IPSBAR + 0x00_00BC (CSAR5); IPSBAR + 0x00_00C8 (CSAR6); IPSBAR + 0x00_00D4 (CSAR7)															

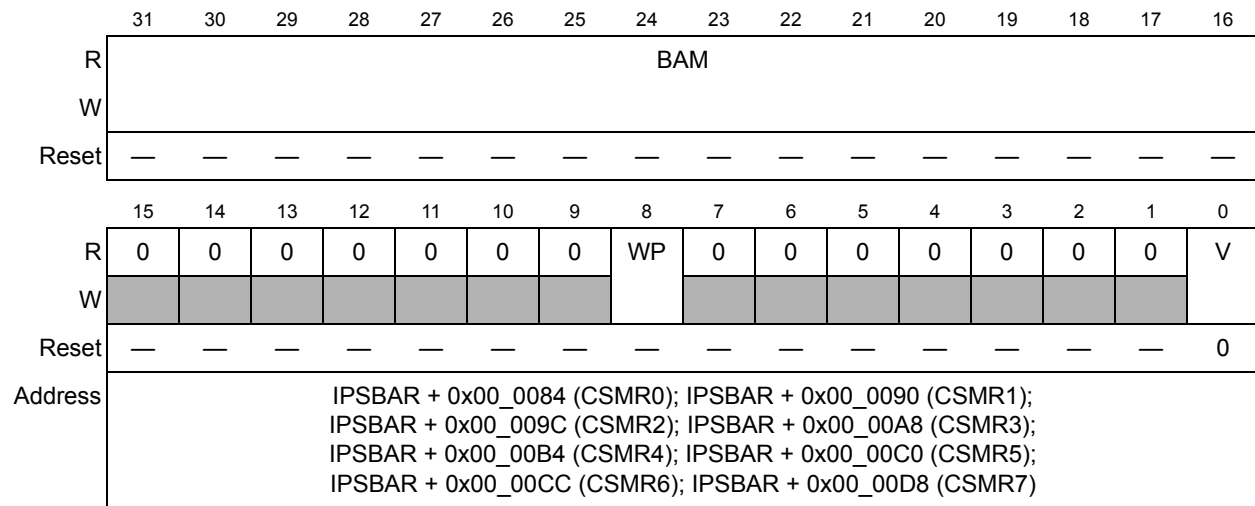
**Figure 15-4. Chip Select Address Registers (CSAR<sub>n</sub>)**

**Table 15-5. CSAR<sub>n</sub> Field Description**

Bits	Name	Description
15–0	BA	Base address. Defines the base address for memory dedicated to chip select $\overline{CS}[7:0]$ . BA is compared to bits 31–16 on the internal address bus to determine if chip select memory is being accessed.

### 15.4.1.2 Chip Select Mask Registers (CSMR0–CSMR7)

The CSMRs, [Figure 15-5](#), are used to specify the address masks for the respective chip selects.

**Figure 15-5. Chip Select Mask Registers (CSMR<sub>n</sub>)****Table 15-6. CSMR<sub>n</sub> Field Descriptions**

Bits	Name	Description
31–16	BAM	Base address mask. Defines the chip select block by masking address bits. Setting a BAM bit causes the corresponding CSAR bit to be ignored in the decode. 0 Corresponding address bit is used in chip select decode. 1 Corresponding address bit is a don't care in chip select decode. The block size for $\overline{CS}[7:0]$ is $2^n$ where $n = (\text{number of bits set in respective CSMR[BAM]}) + 16$ . For example, if $\text{CSAR0} = 0x0000$ and $\text{CSMR0[BAM]} = 0x0001$ , $\overline{CS0}$ addresses a 128-Kbyte ( $2^{17}$ byte) range from $0x0000$ – $0x1\_FFFF$ . Likewise, for $\overline{CS0}$ to access 32 Mbytes ( $2^{25}$ bytes) of address space starting at location $0x0000$ , and for $\overline{CS1}$ to access 16 Mbytes ( $2^{24}$ bytes) of address space starting after the $\overline{CS0}$ space, then $\text{CSAR0} = 0x0000$ , $\text{CSMR0[BAM]} = 0x01FF$ , $\text{CSAR1} = 0x0200$ , and $\text{CSMR1[BAM]} = 0x00FF$ .
8	WP	Write protect. Controls write accesses to the address range in the corresponding CSAR. Attempting to write to the range of addresses for which $\text{CSAR}_n[\text{WP}] = 1$ results in the appropriate chip select not being selected and an access error exception will occur. 0 Both read and write accesses are allowed. 1 Only read accesses are allowed.

**Table 15-6. CSMR<sub>n</sub> Field Descriptions (Continued)**

Bits	Name	Description
7–1	—	Reserved, should be cleared.
0	V	Valid bit. Indicates whether the corresponding CSAR, CSMR, and CSCR contents are valid. Programmed chip selects do not assert until V is set (except for CS <sub>0</sub> , which acts as the global chip select). Reset clears each CSMR <sub>n</sub> [V]. 0 Chip select invalid 1 Chip select valid

### 15.4.1.3 Chip Select Control Registers (CSCR0–CSCR7)

Each CSCR, shown in [Figure 15-6](#), controls the auto-acknowledge, port size, burst capability, and activation of each chip select. Note that to support the external boot chip select, CS<sub>0</sub>, the CSCR<sub>0</sub> reset values differ from the other CSCRs. CS<sub>0</sub> allows address decoding for boot ROM before system initialization.

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	SRWS		IWS				0	AA	PS		BEM	BSTR	BSTW	SWWS		
W																
Reset: CSCR0	0	0	1	1	1	1	0	1	1	0	0	0	0	0	0	0
Reset: Other CSCRs	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Address	IPSBAR + 0x00_008A (CSCR0); IPSBAR + 0x00_0096 (CSCR1); IPSBAR + 0x00_00A2 (CSCR2); IPSBAR + 0x00_00AE (CSCR3); IPSBAR + 0x00_00BA (CSCR4); IPSBAR + 0x00_00C6 (CSCR5); IPSBAR + 0x00_00D2 (CSCR6); IPSBAR + 0x00_00DE (CSCR7)															

**Figure 15-6. Chip Select Control Registers (CSCR<sub>n</sub>)****Table 15-7. CSCR<sub>n</sub> Field Descriptions**

Bits	Name	Description
15–14	SRWS	Secondary Read Wait States. The number of wait states applied to all reads <i>after</i> the initial one if properly enabled (SRWS is non-zero and CSCR[AA] = 1). The default value of this field is secondary read wait states disabled. See <a href="#">Section 15.3.2, “Enhanced Wait State Operation,”</a> for timing diagrams. 00 Secondary read wait states are disabled. Use CSCR[IWS] for all accesses. 01 0 wait states for the secondary read accesses 10 1 wait state for the secondary read accesses 11 2 wait states for the secondary read accesses
13–10	IWS	Initial Wait States. The number of wait states inserted before an internal transfer acknowledge is generated (WS = 0 inserts zero wait states, WS = 0xF inserts 15 wait states). If AA = 0, TA must be asserted by the external system regardless of the number of wait states generated. In that case, the external transfer acknowledge ends the cycle. An external TA supercedes the generation of an internal TA.
9	—	Reserved, should be cleared.

**Table 15-7. CSCR<sub>n</sub> Field Descriptions (Continued)**

Bits	Name	Description
8	AA	Auto-acknowledge enable. Determines the assertion of the internal transfer acknowledge for accesses specified by the chip select address. 0 No internal $\overline{TA}$ is asserted. Cycle is terminated externally. 1 Internal $\overline{TA}$ is asserted as specified by WS. Note that if AA = 1 for a corresponding $\overline{CS}_n$ and the external system asserts an external $\overline{TA}$ before the wait-state countdown asserts the internal $\overline{TA}$ , the cycle is terminated. Burst cycles increment the address bus between each internal termination.
7–6	PS	Port size. Specifies the width of the data associated with each chip select. It determines where data is driven during write cycles and where data is sampled during read cycles. See Section 15.3.1.1. 00 Reserved 01 8-bit port size. Valid data sampled and driven on D[31:24] 1x 16-bit port size. Valid data sampled and driven on D[31:16]
5	BEM	Byte enable mode. Specifies the byte enable operation. Certain SRAMs have byte enables that must be asserted during reads as well as writes. BEM can be set in the relevant CSCR to provide the appropriate mode of byte enable in support of these SRAMs. 0 $\overline{BS}$ is not asserted for read. $\overline{BS}$ is asserted for data write only. 1 $\overline{BS}$ is asserted for read and write accesses.
4	BSTR	Burst read enable. Specifies whether burst reads are used for memory associated with each $\overline{CS}_n$ . 0 Data exceeding the specified port size is broken into individual, port-sized non-burst reads. For example, a longword read from an 8-bit port is broken into four 8-bit reads. 1 Enables data burst reads larger than the specified port size, including longword reads from 8- and 16-bit ports, word reads from 8-bit ports, and line reads from 8- and 16-bit ports.
3	BSTW	Burst write enable. Specifies whether burst writes are used for memory associated with each $\overline{CS}_n$ . 0 Break data larger than the specified port size into individual port-sized, non-burst writes. For example, a longword write to an 8-bit port takes four byte writes. 1 Enables burst write of data larger than the specified port size, including longword writes to 8 and 16-bit ports, word writes to 8-bit ports and line writes to 8- and 16-bit ports.
2–0	SWWS	Secondary write wait states. The number of wait states applied to all writes <i>after</i> the initial one if properly enabled (SWWS is non-zero and CSCR[AA] = 1). The default for this field is secondary write wait states disabled. See Section 15.3.2, “Enhanced Wait State Operation,” for timing diagrams. This field is encoded as:  000 Secondary write wait states are disabled. Use CSCR[IWS] for all accesses. 001 0 wait states for the secondary write accesses 010 1 wait state for the secondary write accesses 011 2 wait states for the secondary write accesses 100 3 wait states for the secondary write accesses 101 4 wait states for the secondary write accesses 110 5 wait states for the secondary write accesses 111 6 wait states for the secondary write accesses

## 15.5 Code Example

```

CSAR0 EQU IPSBARx+0x080 ;Chip select 0 address register
CSMR0 EQU IPSBARx+0x084 ;Chip select 0 mask register
CSCR0 EQU IPSBARx+0x08A ;Chip select 0 control register

CSAR1 EQU IPSBARx+0x08C ;Chip select 1 address register
CSMR1 EQU IPSBARx+0x090 ;Chip select 1 mask register
CSCR1 EQU IPSBARx+0x096 ;Chip select 1 control register

CSAR2 EQU IPSBARx+0x098 ;Chip select 2 address register
CSMR2 EQU IPSBARx+0x09C ;Chip select 2 mask register
CSCR2 EQU IPSBARx+0x0A2 ;Chip select 2 control register

CSAR3 EQU IPSBARx+0x0A4 ;Chip select 3 address register
CSMR3 EQU IPSBARx+0x0A8 ;Chip select 3 mask register
CSCR3 EQU IPSBARx+0x0AE ;Chip select 3 control register

CSAR4 EQU IPSBARx+0x0B0 ;Chip select 4 address register
CSAR4 EQU IPSBARx+0x0B4 ;Chip select 4 mask register
CSMR4 EQU IPSBARx+0x0BA ;Chip select 4 control register

CSAR5 EQU IPSBARx+0x0BC ;Chip select 5 address register
CSMR5 EQU IPSBARx+0x0C0 ;Chip select 5 mask register
CSCR5 EQU IPSBARx+0x0C6 ;Chip select 5 control register

CSAR6 EQU IPSBARx+0x0C8 ;Chip select 6 address register
CSMR6 EQU IPSBARx+0x0CC ;Chip select 6 mask register
CSCR6 EQU IPSBARx+0x0D2 ;Chip select 6 control register

CSAR7 EQU IPSBARx+0x0D4 ;Chip select 7 address register
CSMR7 EQU IPSBARx+0x0D8 ;Chip select 7 mask register
CSCR7 EQU IPSBARx+0x0DE ;Chip select 7 control register

; All other chip selects should be programmed and made valid before global
; chip select is de-activated by validating CS0

; Program Chip Select 3 Registers
move.w #0x0040,D0 ;CSAR3 base address 0x00400000
move.w D0,CSAR3

move.w #0x00A0,D0 ;CSCR3 = no wait states, AA=0, PS=16-bit, BEM=1,
move.w D0,CSCR3 ;BSTR=0, BSTW=0

move.l #0x001F0101,D0;Address range from 0x00400000 to 0x005FFFFFF
move.l D0,CSMR3 ;WP,V=1; SC,UC=0

; Program Chip Select 2 Registers

move.w #0x0020,D0 ;CSAR2 base address 0x00200000 (to 0x003FFFFFF)
move.w D0,CSAR2

move.w #0x0538,D0 ;CSCR2 = 1 wait state, AA=1, PS=32-bit, BEM=1,
move.w D0,CSCR2 ;BSTR=1, BSTW=1

move.l #0x001F0001,D0 ;Address range from 0x00200000 to 0x003FFFFFF
move.l D0,CSMR2 ;W=0; V=1

```

## Chip Select Module

; Program Chip Select 1 Registers

```
move.w #0x0000,D0      ;CSAR1 base addresses 0x00000000 (to 0x001FFFFFF)
move.w D0,CSAR1        ;and 0x80000000 (to 0x801FFFFFF)

move.w #0x09B0,D0      ;CSCR1 = 2 wait states, AA=1, PS=16-bit, BEM=1,
move.w D0,CSCR1        ;BSTR=1, BSTW=0

move.l #0x801F0001,D0   ;Address range from 0x00000000 to 0x001FFFFFF and
move.l D0,CSMR1        ;0x80000000 to 0x801FFFFFF
                        ;WP=0, V=1
```

; Program Chip Select 0 Registers

```
move.w #0x0080,D0      ;CSAR0 base address 0x00800000 (to 0x009FFFFFF)
move.w D0,CSAR0

move.w #0x0D80,D0      ;CSCR0 = three wait states, AA=1, PS=16-bit, BEM=0,
move.w D0,CSCR0        ;BSTR=0, BSTW=0
```

; Program Chip Select 0 Mask Register (validate chip selects)

```
move.l #0x001F0001,D0   ;Address range from 0x00800000 to 0x009FFFFFF
move.l D0,CSMR0        ;WP=0; V=1
```

# Chapter 16

## External Interface Module (EIM)

### 16.1 Introduction

This chapter describes data-transfer operations, error conditions, and reset operations. [Chapter 17, “SDRAM Controller \(SDRAMC\),”](#) describes DRAM cycles.

#### NOTE

Unless otherwise noted, in this chapter, “clock” refers to the CLKOUT used for the bus ( $f_{\text{sys}}/2$ ).

#### NOTE

The timing diagrams within this chapter show 32 address lines, A[31:0]. However, only the lowest 24 address signals are available externally on the MCF5275 device, A[23:0].

#### 16.1.1 Features

The following list summarizes bus operation features:

- Up to 24 bits of address and 16 bits of data
- Access 8- and 16-bit data port sizes
- Generates byte, word, longword, and line-size transfers
- Burst and burst-inhibited transfer support
- Optional internal termination for external bus cycles
- Enhanced secondary wait state

### 16.2 Bus and Control Signals

[Table 16-1](#) summarizes MCF5275 bus signals described in [Chapter 2, “Signal Descriptions.”](#)

**Table 16-1. ColdFire Bus Signal Summary**

Signal Name	Description	I/O	CLKOUT Edge
A[23:0]	Address bus	O	Rising
$\overline{\text{BS}}[3:2]^1$	Byte selects	O	Falling
$\overline{\text{CS}}[7:0]^1$	Chip selects	O	Falling
D[31:16]	Data bus	I/O	Rising
$\overline{\text{OE}}^1$	Output enable	O	Falling

**Table 16-1. ColdFire Bus Signal Summary (Continued)**

Signal Name	Description	I/O	CLKOUT Edge
R/ $\overline{W}$	Read/write	O	Rising
TSIZ[1:0]	Transfer size	O	Rising
$\overline{TA}$	Transfer acknowledge	I	Rising
$\overline{TIP}$	Transfer in progress	O	Rising
$\overline{TS}$	Transfer start	O	Rising

<sup>1</sup> These signals change after the falling edge. In the *Electrical Specifications*, these signals are specified off of the rising edge because CLKIN is squared up internally.

Table 16-2 explains the transfer size encoding of the TSIZ[1:0] signals.

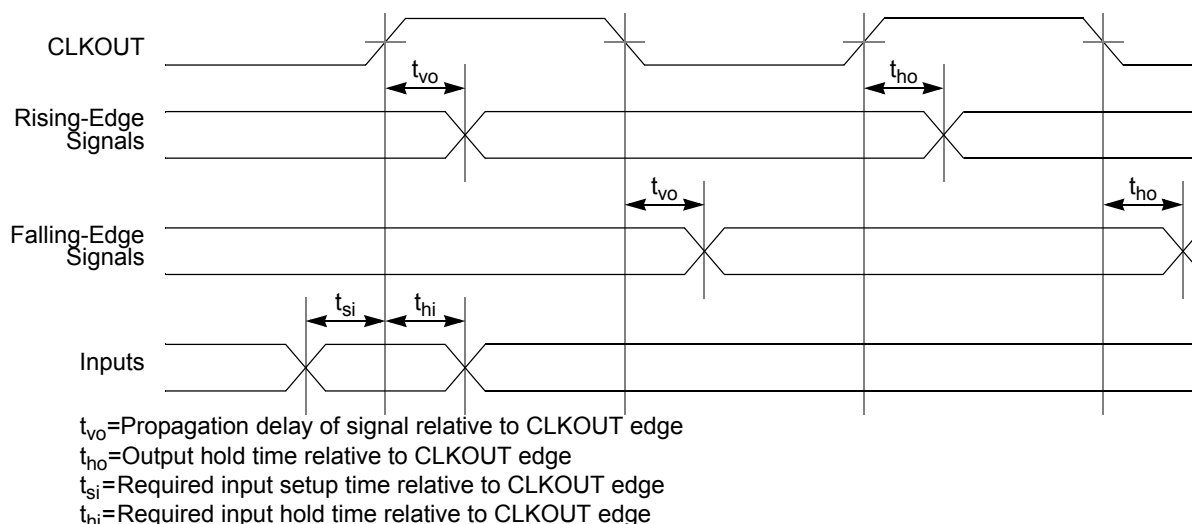
**Table 16-2. Transfer Size Encoding**

TSIZ[1:0]	Transfer Size
00	Longword
01	Byte
10	Word
11	16-byte line

## 16.3 Bus Characteristics

The MCF5275 uses its internal bus clock to generate CLKOUT (see [Chapter 7, “Clock Module”](#)). Therefore, the external bus operates at the same speed as the bus clock rate of  $f_{\text{sys}}/2$ , where all bus operations are synchronous to the rising edge of CLKOUT, and some of the bus control signals ( $\overline{BS}$ ,  $\overline{OE}$ , and  $\overline{CSn}$ ) are synchronous to the falling edge, shown in [Figure 16-1](#). Bus characteristics may differ somewhat for interfacing with external DRAM.





**Figure 16-1. Signal Relationship to CLKOUT for Non-DRAM Access**

## 16.4 Bus Errors

Attempting to write to a range of addresses that is write protected ( $CSAR_n[WP] = 1$ ) will not assert the corresponding chip select. Also, an access error exception will occur. See [Section 15.4.1.2, “Chip Select Mask Registers \(CSMR0–CSMR7\)”](#) and [Section 3.7, “Processor Exceptions”](#) for more details.

## 16.5 Data Transfer Operation

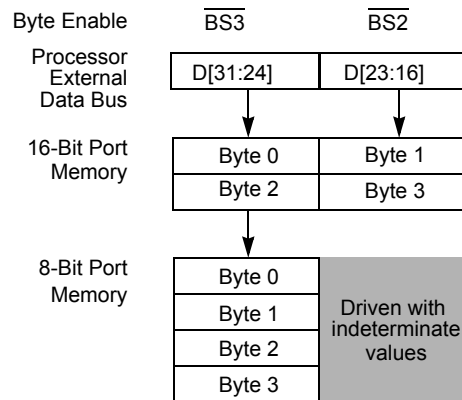
Data transfers between the MCF5275 and other devices involve the following signals:

- Address bus ( $A[23:0]$ )
- Data bus ( $D[31:16]$ )
- Control signals ( $\overline{TS}$  and  $\overline{TA}$ )
- $\overline{CS}_n$ ,  $\overline{OE}$ ,  $\overline{BS}_n$
- Attribute signals ( $R/\overline{W}$ ,  $TSIZ$ , and  $\overline{TIP}$ )

The address bus, write data,  $\overline{TS}$ , and all attribute signals change on the rising edge of CLKOUT. Read data is latched into the MCF5275 on the rising edge of CLKOUT.

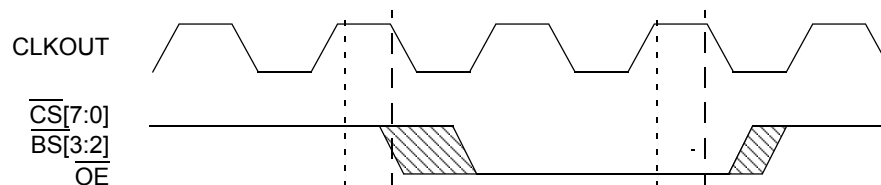
The MCF5275 bus supports byte, word, and longword operand transfers and allows accesses to 8- and 16-bit data ports. Aspects of the transfer, such as the port size, the number of wait states for the external slave being accessed, and whether internal transfer termination is enabled, can be programmed in the chip-select control registers (CSCRs) and the SDRAM base address registers (SDBARs).

Figure 16-2 shows the byte lanes that external memory should be connected to and the sequential transfers if a longword is transferred for two port sizes. For example, an 8-bit memory should be connected to D[31:24] ( $\overline{BS3}$ ). A longword transfer takes four transfers on D[31:24], starting with the MSB and going to the LSB.



**Figure 16-2. Connections for External Memory Port Sizes**

The timing relationship of chip selects ( $\overline{CS}[7:0]$ ), byte selects ( $\overline{BS}[3:2]$ ), and output enable ( $\overline{OE}$ ) with respect to CLKOUT is similar in that all transitions occur during the low phase of CLKOUT. However, due to differences in on-chip signal routing, signals may not assert simultaneously.



**Figure 16-3. Chip-Select Module Output Timing Diagram**

### 16.5.1 Bus Cycle Execution

When a bus cycle is initiated, the MCF5275 first compares the address of that bus cycle with the base address and mask configurations programmed for chip selects 0–7 (configured in CSCR0–CSCR7) and DRAM block 0 and 1 address and control registers (configured in DACR0 and DACR1). If the driven address compares with one of the programmed chip selects or DRAM blocks, the appropriate chip select is asserted or the DRAM block is selected using the specifications programmed by the user in the respective configuration register. Otherwise, the following occurs:

- If the address and attributes do not match in CSCR or DACR, the MCF5275 runs an external burst-inhibited bus cycle with a default of external termination on a 16-bit port.
- Should an address and attribute match in multiple CSCRs, the matching chip-select signals are driven; however, the MCF5275 runs an external burst-inhibited bus cycle with external termination on a 16-bit port.

- Should an address and attribute match both DACRs or a DACR and a CSCR, the operation is undefined.

Table 16-3 shows the type of access as a function of match in the CSCRs and DACRs.

**Table 16-3. Accesses by Matches in CSCRs and DACRs**

Number of CSCR Matches	Number of DACR Matches	Type of Access
0	0	External
1	0	Defined by CSCR
Multiple	0	External, burst-inhibited, 16-bit
0	1	Defined by DACRs
1	1	Undefined
Multiple	1	Undefined
0	Multiple	Undefined
1	Multiple	Undefined
Multiple	Multiple	Undefined

Basic operation of the MCF5275 bus is a three-clock bus cycle.

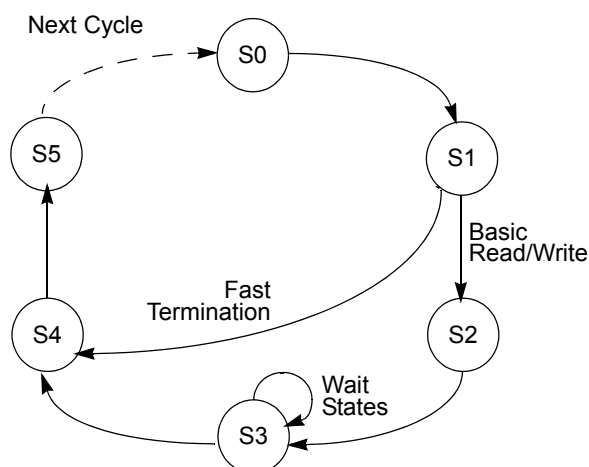
1. During the first clock, the address, attributes, and  $\overline{TS}$  are driven.
2. Data and  $\overline{TA}$  are sampled during the second clock of a bus-read cycle. During a read, the external device provides data and is sampled at the rising edge at the end of the second bus clock. This data is concurrent with  $\overline{TA}$ , which is also sampled at the rising edge of the clock.

During a write, the ColdFire device drives data from the rising clock edge at the end of the first clock to the rising clock edge at the end of the bus cycle. Wait states can be added between the first and second clocks by delaying the assertion of  $\overline{TA}$ .  $\overline{TA}$  can be configured to be generated internally through the CSCRs. If  $\overline{TA}$  is not generated internally, the system must provide it externally.

3. The last clock of the bus cycle uses what would be an idle clock between cycles to provide hold time for address, attributes and write data. Figure 16-6 and Figure 16-8 show the basic read and write operations.

## 16.5.2 Data Transfer Cycle States

The data transfer operation in the MCF5275 is controlled by an on-chip state machine. Each bus clock cycle is divided into two states. Even states occur when CLKOUT is high and odd states occur when CLKOUT is low. The state transition diagram for basic and fast termination read and write cycles are shown in Figure 16-4.



**Figure 16-4. Data Transfer State Transition Diagram**

Table 16-4 describes the states as they appear in subsequent timing diagrams.

**Table 16-4. Bus Cycle States**

State	Cycle	CLKOUT	Description
S0	All	High	The read or write cycle is initiated in S0. On the rising edge of CLKOUT, the MCF5275 places a valid address on the address bus and drives $\overline{R/\overline{W}}$ high for a read and low for a write, if it is not already in the appropriate state. The MCF5275 asserts $\overline{TIP}$ , $TSIZ[1:0]$ , and $\overline{TS}$ on the rising edge of CLKOUT.
S1	All	Low	The appropriate $\overline{CSn}$ , $\overline{BSn}$ , and $\overline{OE}$ signals assert on the CLKOUT falling edge.
	Fast Termination		$\overline{TA}$ must be asserted during S1. Data is made available by the external device and is sampled on the rising edge of CLKOUT with $\overline{TA}$ asserted.
S2	Read/write (skipped fast termination)	High	$\overline{TS}$ is negated on the rising edge of CLKOUT in S2.
	Write		The data bus is driven out of high impedance as data is placed on the bus on the rising edge of CLKOUT.
S3	Read/write (skipped for fast termination)	Low	The MCF5275 waits for $\overline{TA}$ assertion. If $\overline{TA}$ is not sampled as asserted before the rising edge of CLKOUT at the end of the first clock cycle, the MCF5275 inserts wait states (full clock cycles) until $\overline{TA}$ is sampled as asserted.
	Read		Data is made available by the external device on the falling edge of CLKOUT and is sampled on the rising edge of CLKOUT with $\overline{TA}$ asserted.
S4	All	High	The external device should negate $\overline{TA}$ .
	Read (including fast-termination)		The external device can stop driving data after the rising edge of CLKOUT. However data could be driven through the end of S5.

Table 16-4. Bus Cycle States (Continued)

State	Cycle	CLKOUT	Description
S5	S5	Low	$\overline{CSn}$ , $\overline{BSn}$ , and $\overline{OE}$ are negated on the CLKOUT falling edge of S5. The MCF5275 stops driving address lines and $\overline{R/\overline{W}}$ on the rising edge of CLKOUT, terminating the read or write cycle. At the same time, the MCF5275 negates $\overline{TIP}$ , and $TSIZ[1:0]$ on the rising edge of CLKOUT. Note that the rising edge of CLKOUT may be the start of S0 for the next access cycle.
	Read		The external device stops driving data between S4 and S5.
	Write		The data bus returns to high impedance on the rising edge of CLKOUT. The rising edge of CLKOUT may be the start of S0 for the next access.

**NOTE**

An external device has at most two CLKOUT cycles after the start of S4 to tristate the data bus. This applies to basic read cycles, fast termination cycles, and the last transfer of a burst.

**16.5.3 Read Cycle**

During a read cycle, the MCF5275 receives data from memory or from a peripheral device. Figure 16-5 is a read cycle flowchart.

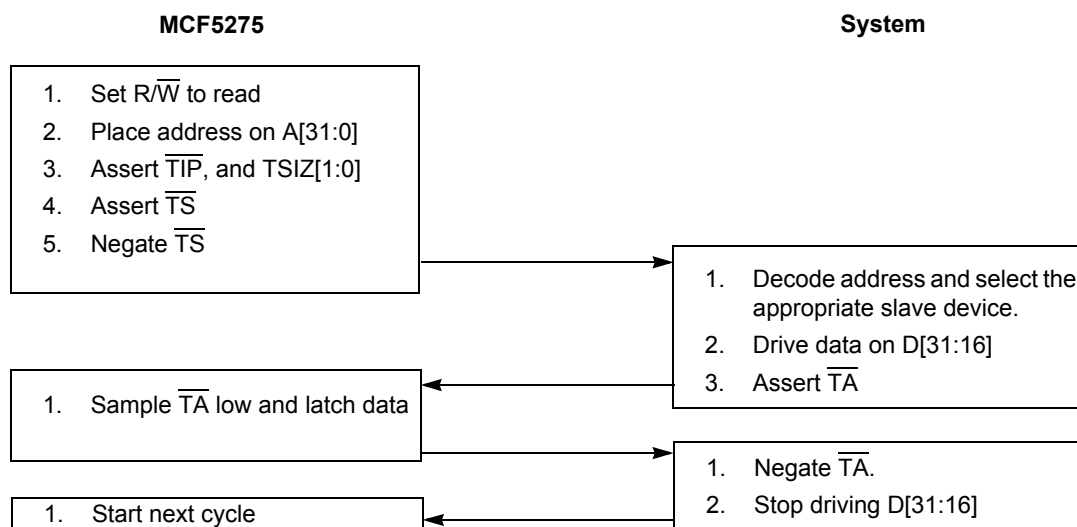
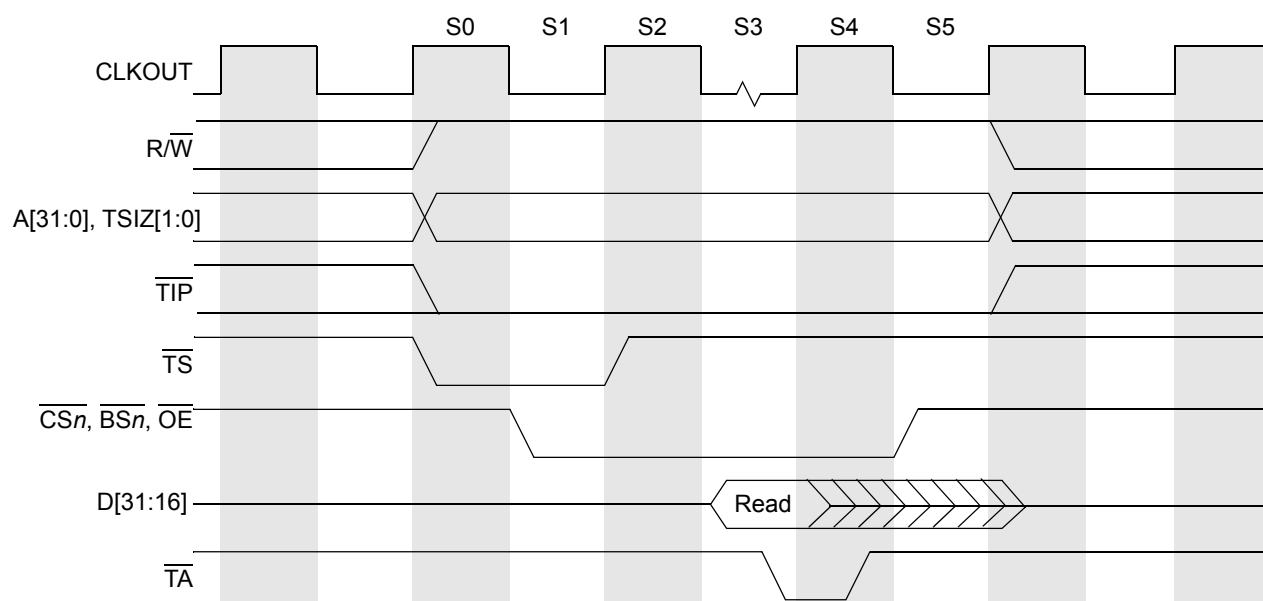


Figure 16-5. Read Cycle Flowchart

The read cycle timing diagram is shown in Figure 16-6.

**NOTE**

In the following timing diagrams,  $\overline{TA}$  waveforms apply for chip selects programmed to enable either internal or external termination.  $\overline{TA}$  assertion should look the same in either case.



**Figure 16-6. Basic Read Bus Cycle**

Note the following characteristics of a basic read:

- In S3, data is made available by the external device on the falling edge of CLKOUT and is sampled on the rising edge of CLKOUT with  $\overline{TA}$  asserted.
- In S4, the external device can stop driving data after the rising edge of CLKOUT. However data could be driven up to S5.
- For a read cycle, the external device stops driving data between S4 and S5.

States are described in [Table 16-4](#).

### 16.5.4 Write Cycle

During a write cycle, the MCF5275 sends data to the memory or to a peripheral device. The write cycle flowchart is shown in [Figure 16-7](#).

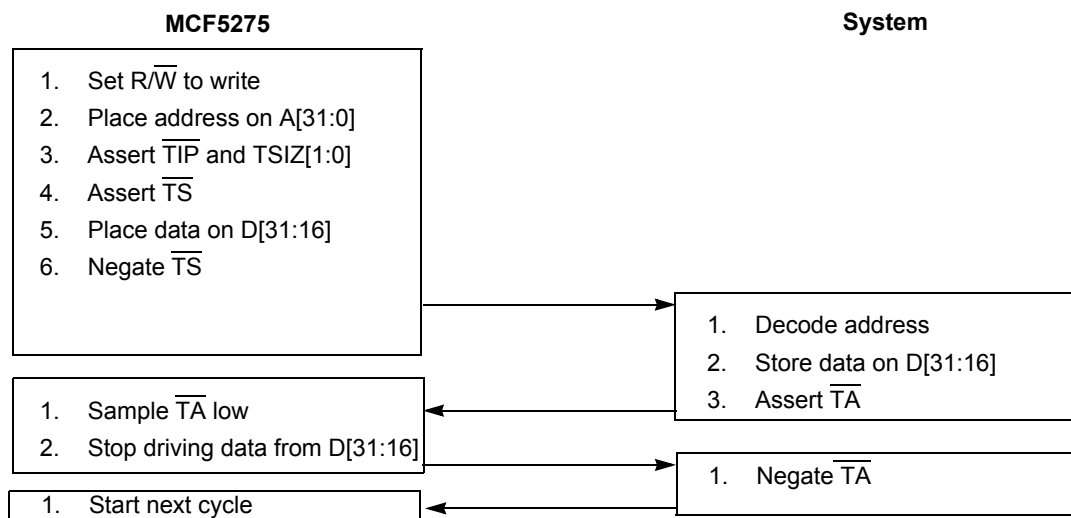


Figure 16-7. Write Cycle Flowchart

The write cycle timing diagram is shown in Figure 16-8.

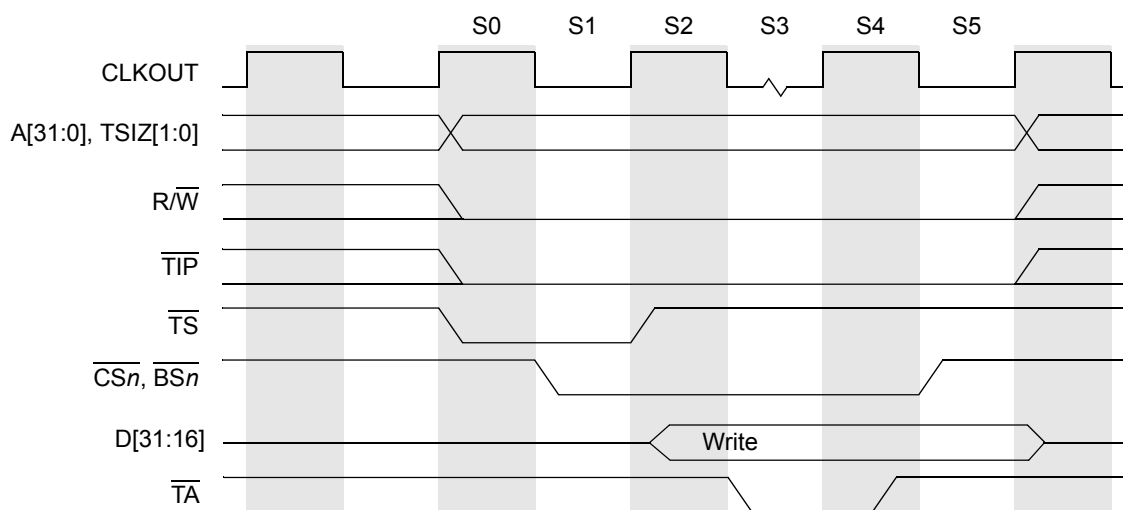


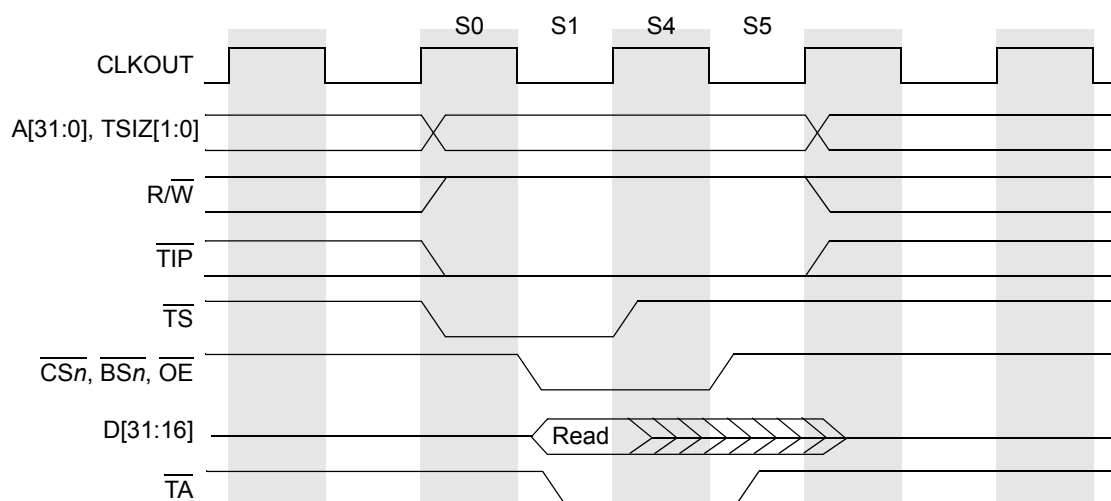
Figure 16-8. Basic Write Bus Cycle

Table 16-4 describes the six states of a basic write cycle.

### 16.5.5 Fast Termination Cycles

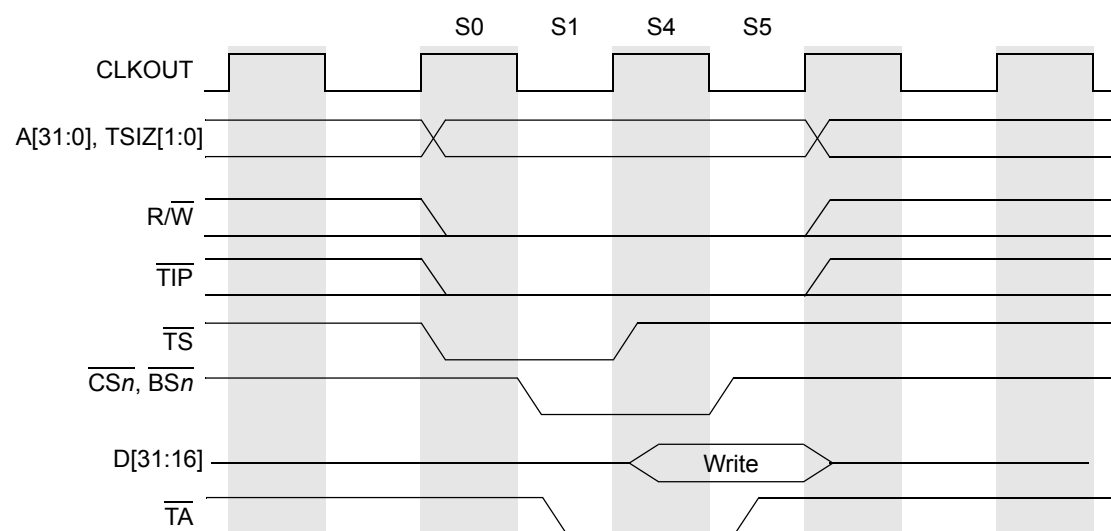
Two clock cycle transfers are supported on the MCF5275 bus. In most cases, this is impractical to use in a system because the termination must take place in the same half-clock during which  $\overline{TS}$  is asserted. As this is atypical, it is not referred to as the zero-wait-state case but is called the fast-termination case. Fast termination cycles occur when the external device or memory asserts  $\overline{TA}$  less than one clock after  $\overline{TS}$  is asserted. This means that the MCF5275 samples  $\overline{TA}$  on the rising

edge of the second cycle of the bus transfer. [Figure 16-9](#) shows a read cycle with fast termination. Note that fast termination cannot be used with internal termination.



**Figure 16-9. Read Cycle with Fast Termination**

[Figure 16-10](#) shows a write cycle with fast termination.

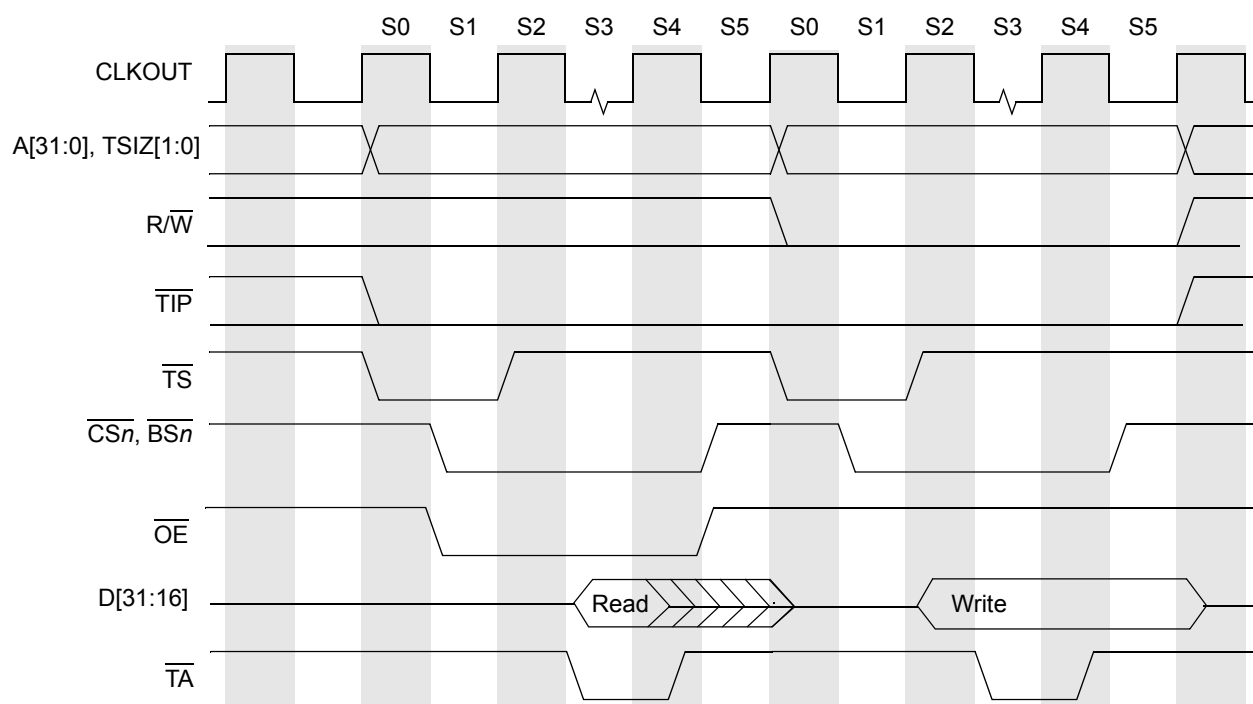


**Figure 16-10. Write Cycle with Fast Termination**

## 16.5.6 Back-to-Back Bus Cycles

The MCF5275 runs back-to-back bus cycles whenever possible. For example, when a longword read is started on a word-size bus, the processor performs two back-to-back word read accesses. Back-to-back accesses are distinguished by the continuous assertion of  $\overline{TIP}$  throughout the cycle. [Figure 16-11](#) shows a read back-to-back with a write.





**Figure 16-11. Back-to-Back Bus Cycles**

Basic read and write cycles are used to show a back-to-back cycle, but there is no restriction as to the type of operations to be placed back to back. The initiation of a back-to-back cycle is not user definable.

### 16.5.7 Burst Cycles

The MCF5275 can be programmed to initiate burst cycles if its transfer size exceeds the size of the port it is transferring to. For example, a word transfer to an 8-bit port would take a 2-byte burst cycle. A line transfer to a 16-bit port would take a 8-word burst cycle.

The MCF5275 bus can support 2-1-1-1 burst cycles to maximize cache performance and optimize DMA transfers. A user can add wait states by delaying termination of the cycle. The initiation of a burst cycle is encoded on the size pins. For burst transfers to smaller port sizes, TSIZ[1:0] indicates the size of the entire transfer. For example, if the MCF5275 writes a longword to an 8-bit port, TSIZ[1:0] = 00 for the first byte transfer and does not change.

The CSCRs can be used to enable bursting for reads, writes, or both. MCF5275 memory space can be declared burst-inhibited for reads and writes by clearing the appropriate CSCRn[BSTR,BSTW]. A line access to a burst-inhibited region first accesses the MCF5275 bus encoded as a line access. The TSIZ[1:0] encoding does not exceed the programmed port size. The address changes if internal termination is used but does not change if external termination is used, as shown in [Figure 16-12](#) and [Figure 16-13](#).

### 16.5.7.1 Line Transfers

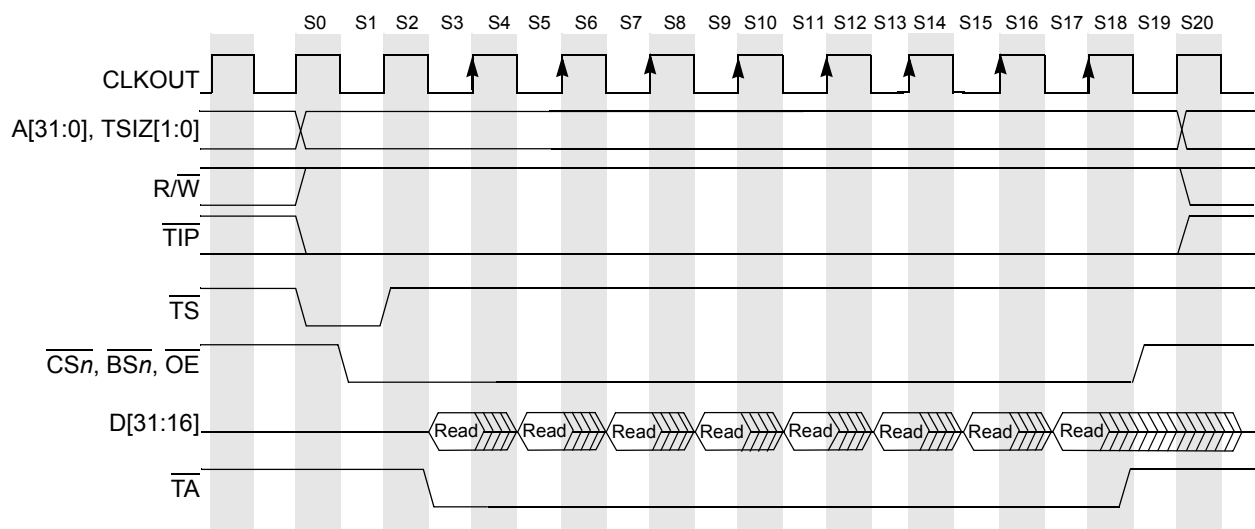
A line is a 16-byte-aligned, 16-byte value. Despite the alignment, a line access may not begin on the aligned address; therefore, the bus interface supports line transfers on multiple address boundaries. Table 16-5 shows allowable patterns for line accesses.

**Table 16-5. Allowable Line Access Patterns**

A[3:2]	Word Accesses
00	0-2-4-6-8-A-C
01	4-6-8-A-C-E-0
10	8-A-C-E-0-2-4
11	C-E-0-2-4-6-8

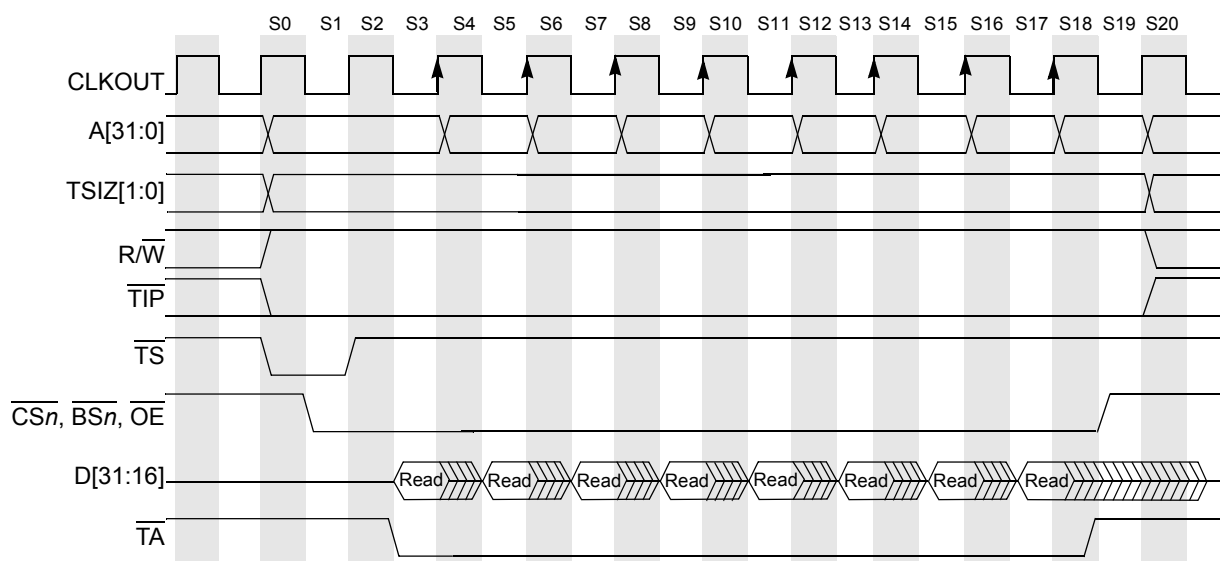
### 16.5.7.2 Line Read Bus Cycles

Figure 16-12 and Figure 16-13 show a line access read with zero wait states. The access starts like a basic read bus cycle with the first data transfer sampled on the rising edge of S4, but the next pipelined burst data is sampled a cycle later on the rising edge of S6. Each subsequent pipelined data burst is single cycle until the last one, which can be held for up to two CLKOUT cycles after  $\overline{TA}$  is asserted. Note that  $\overline{CSn}$  is asserted throughout the burst transfer. This example shows the timing for external termination, which differs from the internal termination example in Figure 16-13 only in that the address lines change only at the beginning (assertion of  $\overline{TS}$  and  $\overline{TIP}$ ) and end (negation of  $\overline{TIP}$ ) of the transfer.



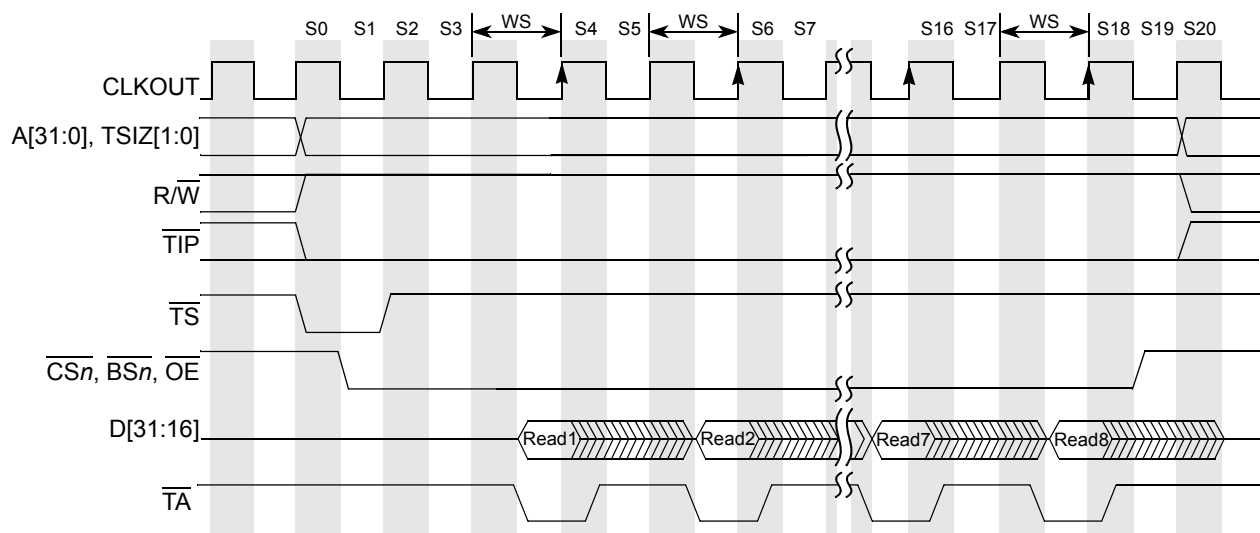
**Figure 16-12. Line Read Burst (2-1-1-1-1-1-1), External Termination**

Figure 16-13 shows timing when internal termination is used.



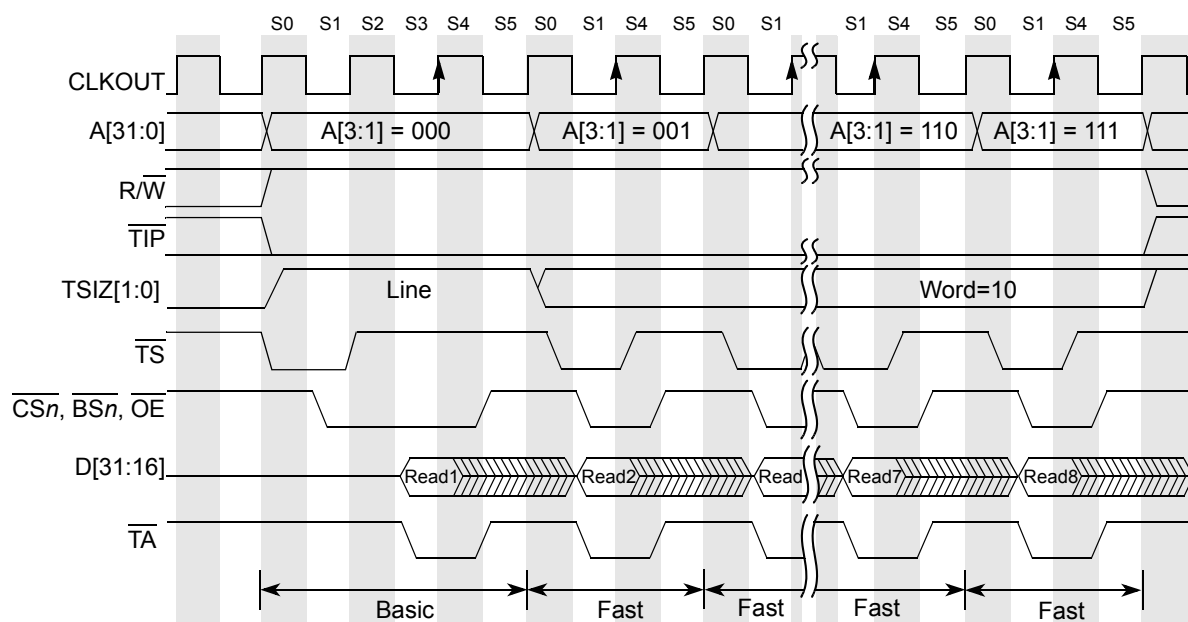
**Figure 16-13. Line Read Burst (2-1-1-1-1-1-1), Internal Termination**

Figure 16-14 shows a line access read with one wait state programmed in  $CSCR_n$  to give the peripheral or memory more time to return read data. This figure follows the same execution as a zero-wait state read burst with the exception of an added wait state.



**Figure 16-14. Line Read Burst (3-2-2-2-2-2-2), External Termination**

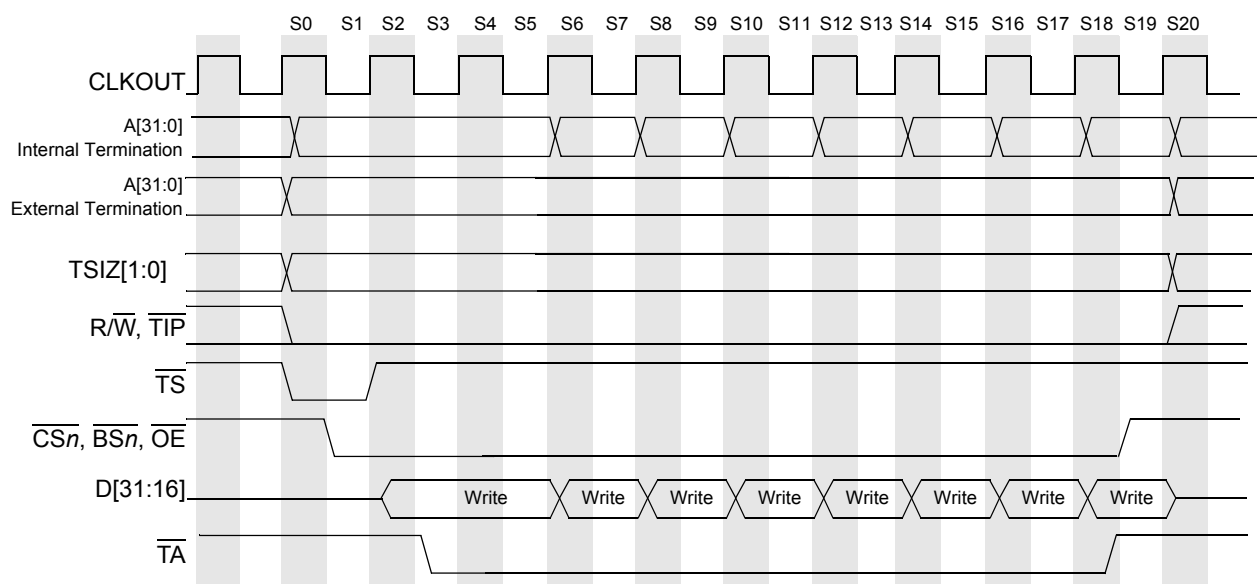
Figure 16-15 shows a burst-inhibited line read access with fast termination. The external device executes a basic read cycle while determining that a line is being transferred. The external device uses fast termination for subsequent transfers.



**Figure 16-15. Line Read Burst-Inhibited, Fast Termination, External Termination**

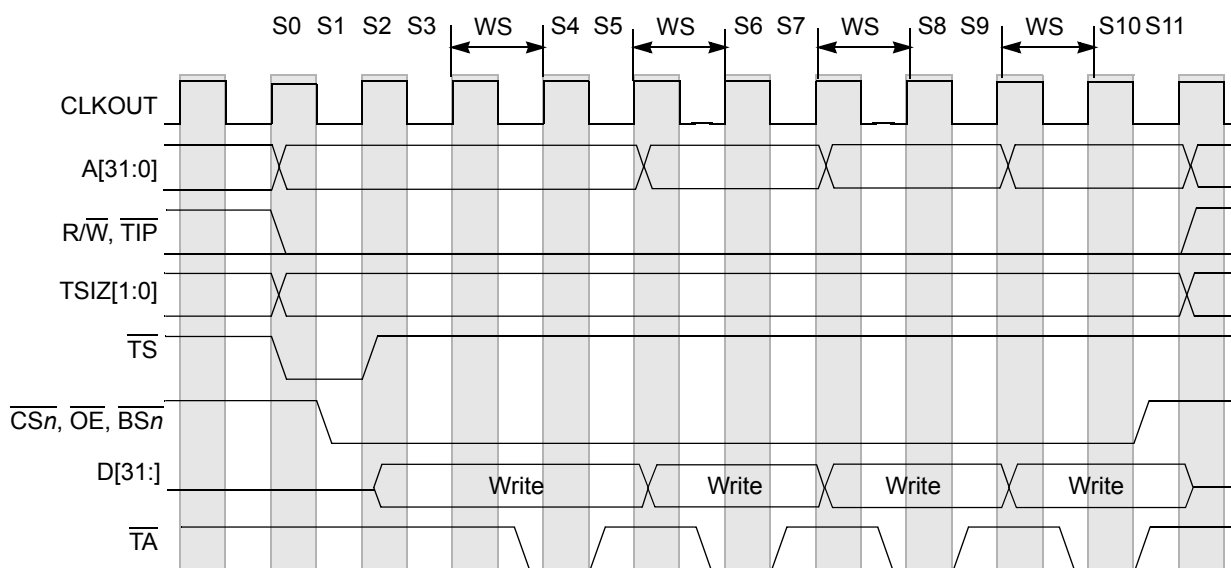
### 16.5.7.3 Line Write Bus Cycles

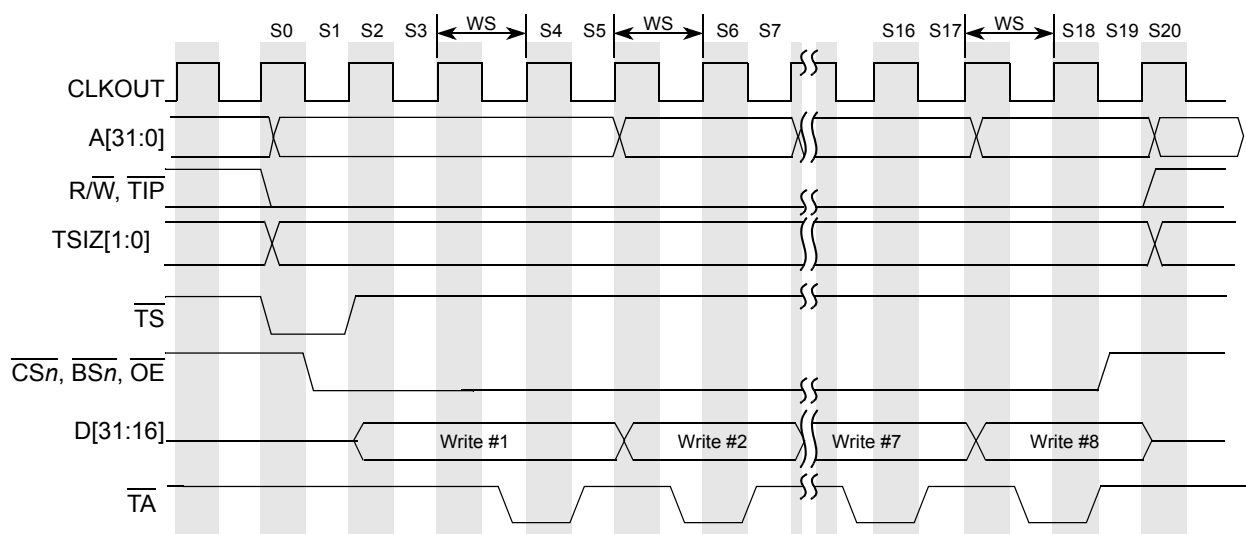
Figure 16-16 shows a line access write with zero wait states. It begins like a basic write bus cycle with data driven one clock after  $\overline{TS}$ . The next pipelined burst data is driven a cycle after the write data is registered (on the rising edge of S6). Each subsequent burst takes a single cycle. Note that as with the line read example in Figure 16-12,  $\overline{CSn}$  remain asserted throughout the burst transfer. This example shows the behavior of the address lines for both internal and external termination. Note that when external termination is used, the address lines change with TSIZ[1:0].



**Figure 16-16. Line Write Burst (2-1-1-1-1-1-1-1), Internal/External Termination**

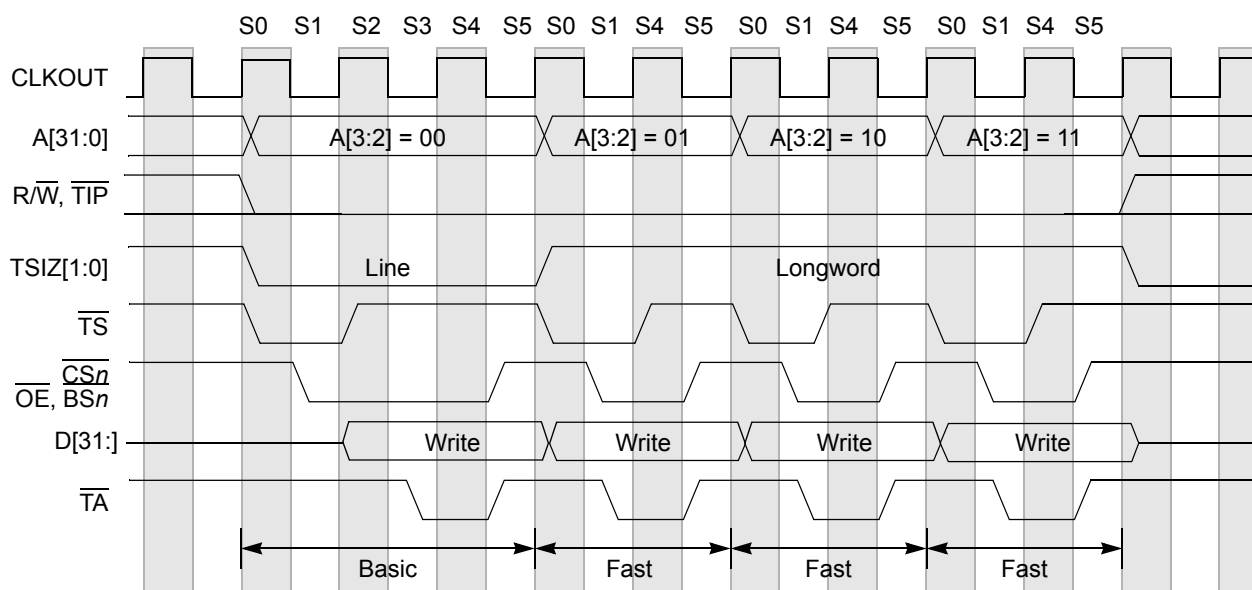
Figure 16-17 shows a line burst write with one wait-state insertion.





**Figure 16-17. Line Write Burst (3-2-2-2-2-2-2-2) with One Wait State**

Figure 16-18 shows a burst-inhibited line write. The external device executes a basic write cycle while determining that a line is being transferred. The external device uses fast termination to end each subsequent transfer.



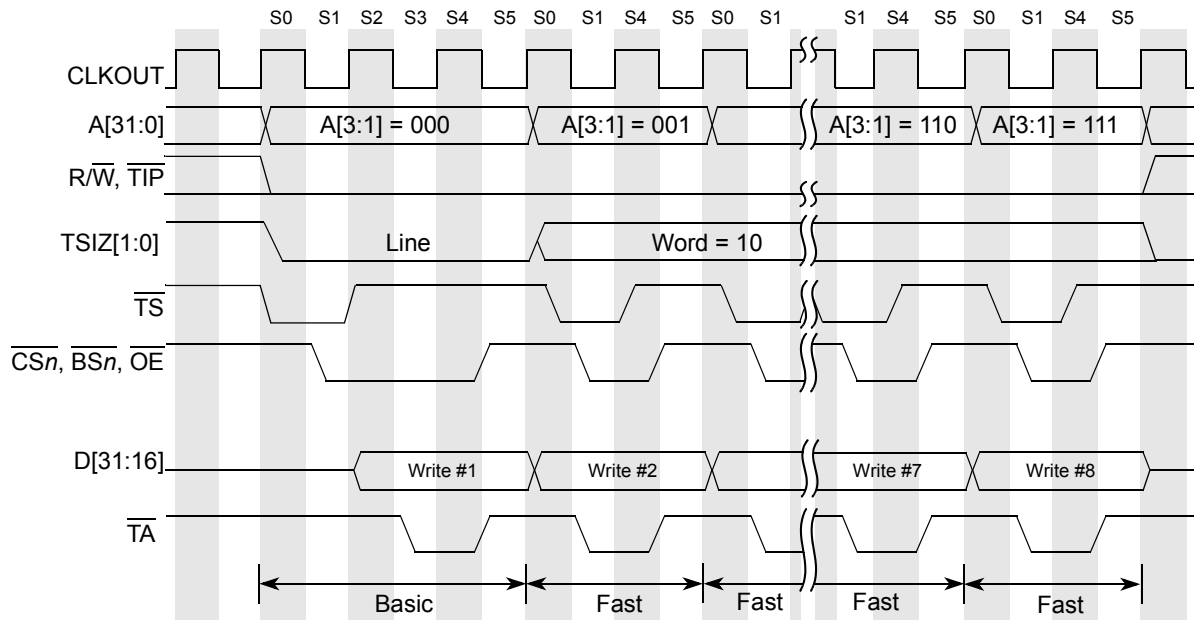


Figure 16-18. Line Write Burst-Inhibited

## 16.6 Secondary Wait State Operation

Refer to [Section 15.3.2, “Enhanced Wait State Operation,”](#) for information and timing diagrams of the secondary wait state operation.

## 16.7 Misaligned Operands

Because operands can reside at any byte boundary, unlike opcodes, they are allowed to be misaligned. A byte operand is properly aligned at any address, a word operand is misaligned at an odd address, and a longword is misaligned at an address not a multiple of four. Although the MCF5275 enforces no alignment restrictions for data operands (including program counter (PC) relative data addressing), additional bus cycles are required for misaligned operands.

Instruction words and extension words (opcodes) must reside on word boundaries. Attempting to prefetch a misaligned instruction word causes an address error exception.

The MCF5275 converts misaligned, cache-inhibited operand accesses to multiple aligned accesses. [Figure 16-19](#) shows the transfer of a longword operand from a byte address to a 32-bit port. In this example, TSIZ[1:0] specify a byte transfer and a byte offset of 0x1. The slave device supplies the byte and acknowledges the data transfer. When the MCF5275 starts the second cycle, TSIZ[1:0] specify a word transfer with a byte offset of 0x2. The next two bytes are transferred in this cycle. In the third cycle, byte 3 is transferred. The byte offset is now 0x0, the port supplies the final byte, and the operation is complete.

	31	24 23	16 15	8 7	0	A[2:0]
Transfer 1	—	Byte 0	—	—	—	001
Transfer 2	—	—	Byte 1	Byte 2	—	010
Transfer 3	Byte 3	—	—	—	—	100

**Figure 16-19. Example of a Misaligned Longword Transfer (32-Bit Port)**

If an operand is cacheable and is misaligned across a cache-line boundary, both lines are loaded into the cache. The example in [Figure 16-20](#) differs from that in [Figure 16-19](#) in that the operand is word-sized and the transfer takes only two bus cycles.

	31	24 23	16 15	8 7	0	A[2:0]
Transfer 1	—	—	—	Byte 0	—	001
Transfer 2	Byte 1	—	—	—	—	100

**Figure 16-20. Example of a Misaligned Word Transfer (32-Bit Port)**



# Chapter 17

## SDRAM Controller (SDRAMC)

### 17.1 Introduction

This chapter describes configuration and operation of the synchronous DRAM (SDRAM) controller. It begins with a general overview and includes a description of signals involved in SDRAM operations. The remainder of the chapter describes the programming model and signal timing, as well as the command set required for synchronous DRAM operations. It also includes examples that the designer can follow to better understand how to configure the SDRAM controller for synchronous operations.

#### 17.1.1 Features

The MCF5275 SDRAM controller contains the following features:

- Supports a glueless interface to DDR SDRAMs
- 16-bit fixed memory port width
- 32-bit internal data bus interface to Coldfire core
- 16 bytes (8 beat x 16-bit) critical word first burst transfer
- Up to 14 row address lines, up to 12 column address lines, maximum of two chip selects. The maximum row bits plus column bits is 24.
- Supported SDRAM devices include: 32, 64, 128, 256, 512Mbit, and 1Gbit per chip select
- Minimum memory configuration of 8 Mbyte—12 bit row address (RA), 8 bit column address (CA), 2 bit bank address (BA) and one chip select
- Supports page mode to maximize the data rate
- Supports sleep mode and self-refresh mode
- Error detect and parity check are not supported

#### 17.1.2 Terminology

The following terminology is used in this chapter:

- **SDRAM block:** Any group of DRAM memories selected by one of the MCF5275 `SD_CS[1:0]` signals. Thus, the MCF5275 can support up to two independent memory blocks. The base address of each block is programmed in the SDRAM base address and mask registers.

- SDRAM bank: An internal partition in an SDRAM device. For example, a 64-Mbit SDRAM component might be configured as four 512K x 32 banks. Banks are selected through the BS[3:2] signals.
- SDRAM: These are RAMs that operate like asynchronous DRAMs but with a synchronous clock, a pipelined, multiple-bank architecture, and a faster speed.
- Double data rate (DDR) SDRAM: This is SDRAM that latches command information on the rising edge of the clock, but data is driven/latched on both the rising and falling edges of the clock rather than on just the rising edge. This doubles data throughput rate without an increase in frequency.

### **17.1.3 Block Diagram**

Below is the SDRAM memory controller block diagram. It illustrates how the module is connected to the internal device, as well as the external interface signals.

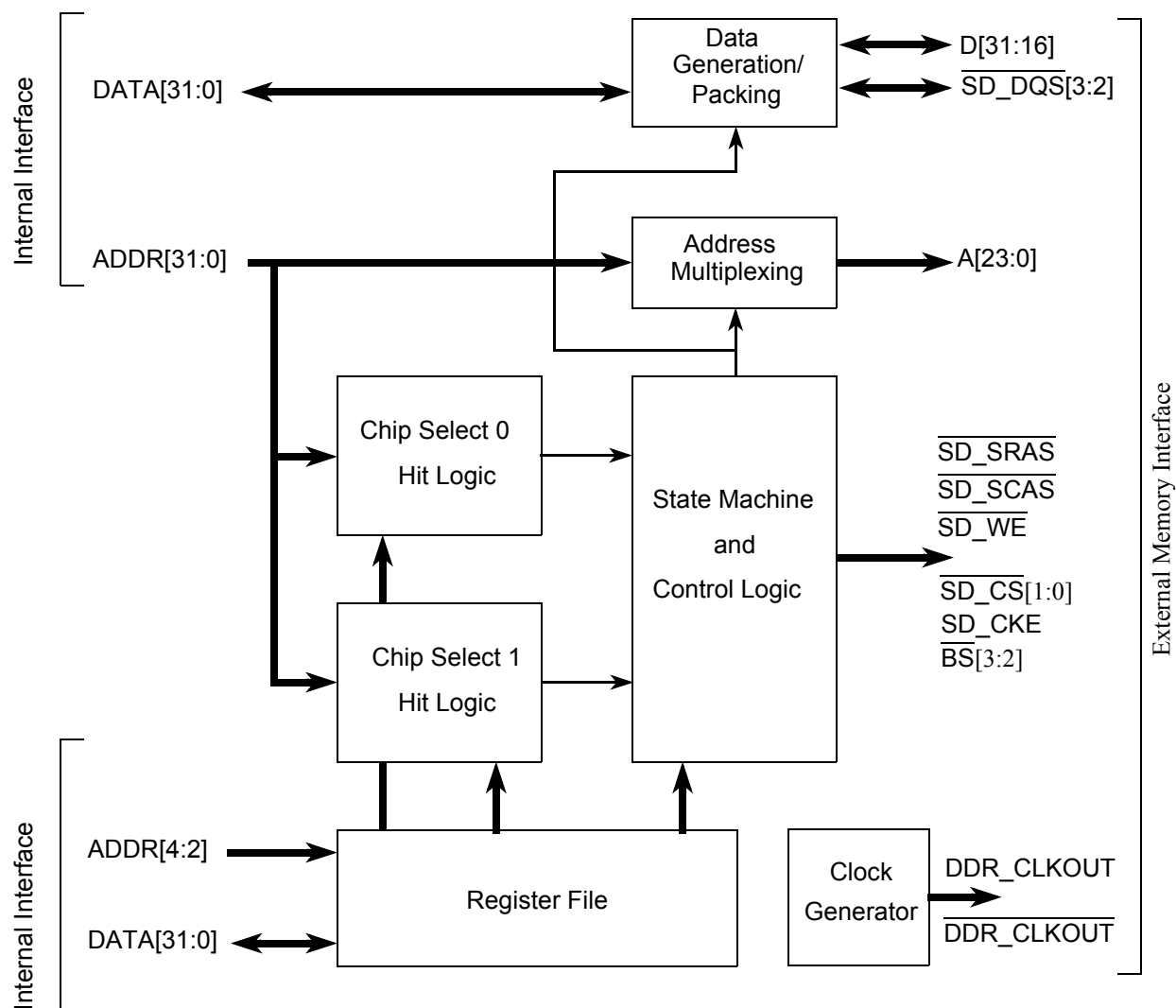


Figure 17-1. SDRAM Controller Block Diagram

## 17.2 External Signal Description

**Table 17-1. SDRAM Controller Signals**

Signal Name	Abbreviation	Function	I/O
Data Bus	D[31:16]	These three-state bidirectional signals provide the general purpose data path between the processor and all other devices. Data is sampled by the MCF5275 on both the rising and falling edge of DDR_CLKOUT	I/O
Bank Selects	A[15:14]	For SDRAM accesses A[15:14] are always used to provide bank select signals to the SDRAM. A[15:14] should be connected to BA[1:0] on the SDRAM device.	O
Address Bus	A[13:11,9:0]	A portion of the 24-bit uni-directional address bus is used for multiplexed row and column addresses during SDRAM bus cycles. The address multiplexing supports up to 128 Mbytes of SDRAM per chip select.	O
SDRAM A10	SD_A10	SDRAM address bit 10 or command.	O
SDRAM Synchronous Row Address Strobe	$\overline{\text{SD\_SRAS}}$	SDRAM synchronous row address strobe.	O
SDRAM Synchronous Column Address Strobe	$\overline{\text{SD\_SCAS}}$	SDRAM synchronous column address strobe.	O
SDRAM Chip Selects	$\overline{\text{SD\_CS}}[1:0]$	These signals interface to the chip select lines of the SDRAMs within a memory block.	O
Byte Strobes	$\overline{\text{BS}}[3:2]$	For SDRAM devices, these signals indicate transfers between SDRAM and the device when driven high. The $\overline{\text{BS}}[3:2]$ should be connected to individual SDRAM DM signals. Note that most SDRAMs associate DM1 with the MSB, in which case BS3 should be connected to the SDRAM's DM1 input.	O
SDRAM Data Strobes	$\overline{\text{SD\_DQS}}[3:2]$	SDRAM byte-lane read/write data strobe signals.	I/O
SDRAM Clock Out	DDR_CLKOUT	This output signal reflects the internal system clock.	O
SDRAM Inverted Clock Out	$\overline{\text{DDR\_CLKOUT}}$	This output signal reflects the inverted internal system clock. used along with DDR_CLKOUT to provide differential clocks for DDR SDRAMs.	O
SDRAM Write Enable	$\overline{\text{SD\_WE}}$	Asserted to signify that a DRAM write cycle is underway. A read cycle is indicated by the deassertion of $\overline{\text{SD\_WE}}$	O
SDRAM Clock Enable	SD_CKE	Deasserts to put the SDRAM into low-power, self-refresh mode.	O

## 17.3 Interface Recommendations

### 17.3.1 Supported Memory Configurations

The SDRAM controller supports up to 14 row addresses and up to 12 column addresses. However, the maximum row and column addresses are not supported at the same time. The number of row and column addresses must be less than or equal to 24. In addition to row/column address lines,

there are always two row bank address bits. Therefore, the greatest possible address space which can be accessed using a single chip select is  $(2^{26}) \times 16$  bit, or 128 Mbytes.

Table 17-2 shows the address multiplexing used by the MCF5275 for different configurations. When the SDRAM controller receives the internal module enable, it latches the internal bus address lines  $\text{addr}[26:1]$  and multiplexes them into row, column and row bank addresses.  $\text{addr}[8:1]$  are always used for  $\text{CA}[7:0]$ ,  $\text{addr}[10:9]$  are always used for  $\text{BA}[1:0]$ , and  $\text{addr}[22:11]$  are always used for  $\text{RA}[11:0]$ .  $\text{addr}[26:23]$  can be used for additional row or column address bits, as needed.

### NOTE

The SDRAMC only supports an external 16-bit data bus. It is not possible to connect a smaller device(s) to only part of the SDRAM's data bus. For example, if 8-bit wide devices are used, then you must use two 8-bit devices connected as a 16-bit port.

**Table 17-2. SDRAM Address Multiplexing**

Device Size	Configuration	Row bit x Col bit x Banks	Number of Chips Used	Total Block Size	SDCR [MUX] Setting	Internal Address						
						26	25	24	23	22-11	10-9	8-1
32 Mbit	4M x 8bit	12 x 8 x 2	2	8 MB	00	—	—	—	—	RA11-0	BA1-0	CA7-0
64 Mbits	4M x 16 bit	12 x 8 x 2	1	8 MB	00	—	—	—	—	RA11-0	BA1-0	CA7-0
	8M x 8bit	12 x 9 x 2	2	16 MB	00	—	—	—	CA8			
		13 x 8 x 2			01	—	—	—	RA12			
	16M x 4bit	12 x 10 x 4	4	32 MB	00	—	—	CA9	CA8			
		13 x 9 x 4			01	—	—	CA8	RA12			
128 Mbits	8M x 16 bit	12 x 9 x 2	1	16 MB	00	—	—	—	CA8	RA11-0	BA1-0	CA7-0
		13 x 8 x 2			01	—	—	—	RA12			
	16M x 8 bit	12 x 10 x 2	2	32 MB	00	—	—	CA9	CA8			
		13 x 9 x 2			01	—	—	CA8	RA12			
		14 x 8 x 2			10	—	—	RA13	RA12			
	32M x 4 bit	12 x 11 x 4	4	64 MB	00	—	CA11	CA9	CA8			
		13 x 10 x 4			01	—	CA9	CA8	RA12			
256 Mbits	16M x 16 bit	12 x 10 x 2	1	32 MB	00	—	—	CA9	CA8	RA11-0	BA1-0	CA7-0
		13 x 9 x 2			01	—	—	CA8	RA12			
		14 x 8 x 2			10	—	—	RA13	RA12			
	32M x 8 bit	12 x 11 x 2	2	64 MB	00	—	CA11	CA9	CA8			
		13 x 10 x 2			01	—	CA9	CA8	RA12			
		14 x 9 x 2			10	—	CA8	RA13	RA12			
	64M x 4 bit	12 x 12 x 4	4	128 MB	00	CA12	CA11	CA9	CA8			
		13 x 11 x 4			01	CA11	CA9	CA8	RA12			

**Table 17-2. SDRAM Address Multiplexing**

Device Size	Configuration	Row bit x Col bit x Banks	Number of Chips Used	Total Block Size	SDCR [MUX] Setting	Internal Address						
						26	25	24	23	22-11	10-9	8-1
512 Mbits	32M x 16 bit	12 x 11 x 2	1	64 MB	00	—	CA11	CA9	CA8	RA11-0	BA1-0	CA7-0
		13 x 10x 2			01	—	CA9	CA8	RA12			
		14 x 9 x 2			10	—	CA8	RA13	RA12			
	64M x 8bit	12 x 12 x 2	2	128 MB	00	CA12	CA11	CA9	CA8			
		13 x 11 x 2			01	CA11	CA9	CA8	RA12			
		14 x 10 x 2			10	CA9	CA8	RA13	RA12			
1 Gbit	64M x 16 bit	12 x 12 x 2	1	128 MB	00	CA12	CA11	CA9	CA8	RA11-0	BA1-0	CA7-0
		13 x 11 x 2			01	CA11	CA9	CA8	RA12			
		14 x 10 x 2			10	CA9	CA8	RA13	RA12			

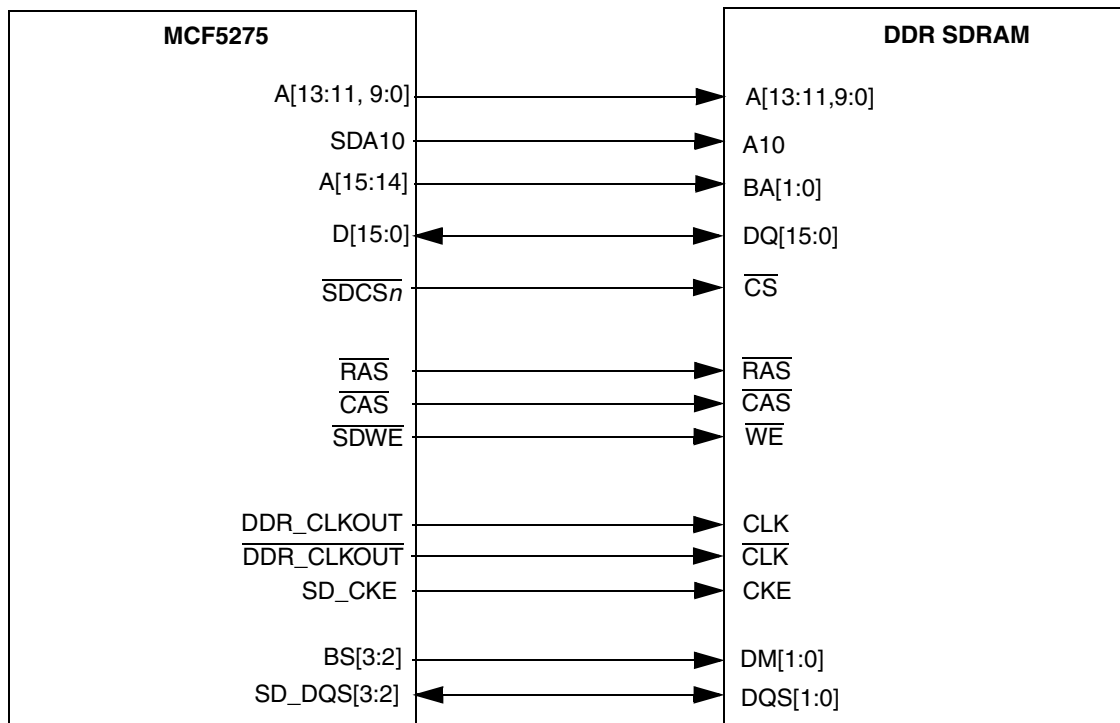
All memory devices of a single chip select block must have the same configuration and row/column address width; however, this is not necessary between different blocks. If mixing different memory organizations in different blocks, the following guidelines will ensure that every block is fully contiguous.

- If all devices' row address width is 12 bits, the column address can be  $\geq 8$  bits.
- If all devices' row address width is 13 bits, the column address can be  $\geq 8$  bits.
- If all devices' column address width is 8 bits, the row address can be  $\geq 11$  bits.
- x8 width memory devices can be mixed (but not in the same space).
- x16 data width memory devices cannot be mixed with any other width.

## 17.3.2 SDRAM Connection Block Diagram

Figure 17-2 shows a block diagram of the connections between the MCF5275 and DDR SDRAM components.

**Figure 17-2. MCF5275 Connections to DDR SDRAM**



## 17.3.3 DDR SDRAM Layout Considerations

Due to the critical timing for DDR SDRAM there are a number of considerations that should be taken into account during PCB layout:

- Minimize overall trace lengths.
- Each SD\_DQS, BS, and DQ group must have identical loading and similar routing to maintain timing integrity.
- Control and clock signals are routed point-to-point.
- Trace length for clock, address, and command signals should match.
- Route DDR signals on layers adjacent to the ground plane
- Use a VREF plane under the SDRAM.
- VREF is decoupled from both SDVDD and VSS.
- To avoid crosstalk keep address and command signals separate from data and data strobes.
- Use different resistor packs for command/address and data/data strobes.

- Use single series, single parallel termination (25  $\Omega$  series, 50  $\Omega$  parallel values are recommended, but standard resistor packs with similar values can be substituted).
- Series termination should be between the MCF5275 and memory, but closest to the processor.
- The parallel termination at end of the signal line (close to the SDRAM).
- 0.1  $\mu$ F decoupling for every termination resistor pack.

### 17.3.3.1 Termination Example

Figure 17-3 shows the recommended termination circuitry for DDR SDRAM signals.

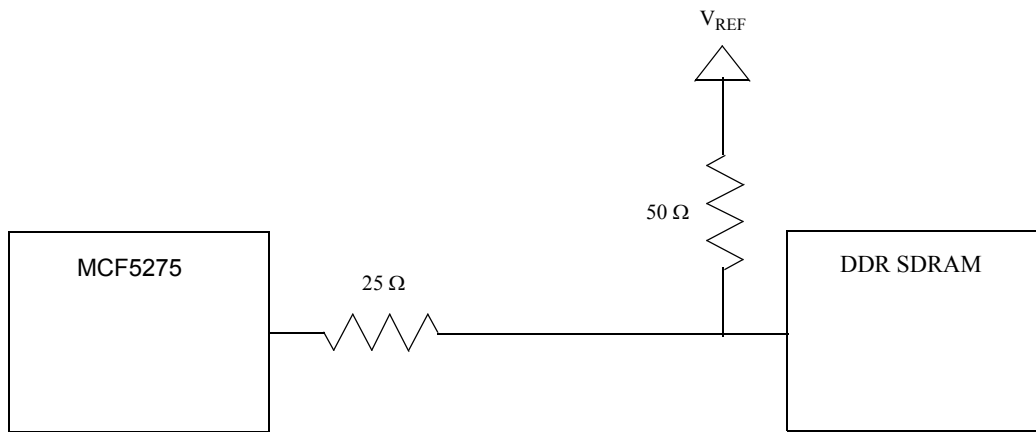


Figure 17-3. MCF5275 DDR SDRAM Termination Circuit

## 17.4 SDRAM Overview

### 17.4.1 SDRAM Commands

When an internal bus master accesses SDRAM address space, the memory controller generates the corresponding SDRAM command. Table 17-3 lists SDRAM commands supported by the memory controller.

Table 17-3. SDRAM Commands

Function	Symbol	CKE	$\overline{\text{CS}}$	$\overline{\text{RAS}}$	$\overline{\text{CAS}}$	$\overline{\text{WE}}$	BA[1:0]	A10	Other A
Command Inhibit	INH	H	H	X	X	X	X	X	X
Read	READ	H	L	H	L	H	V	L	V
Write	WRITE	H	L	H	L	L	V	L	V
Activate	ACTV	H	L	L	H	H	V	V	V
Precharge All Banks	PALL	H	L	L	H	L	X	H	X



**Table 17-3. SDRAM Commands (Continued)**

Function	Symbol	CKE	$\overline{\text{CS}}$	$\overline{\text{RAS}}$	$\overline{\text{CAS}}$	$\overline{\text{WE}}$	BA[1:0]	A10	Other A
Load Mode Register	LMR	H	L	L	L	L	LL	V	V
Load Extended Mode Register	LEMR	H	L	L	L	L	LH	V	V
CBR Auto Refresh	REF	H	L	L	L	H	X	X	X
Self Refresh	SREF	H→L	L	L	L	H	X	X	X
Power Down	PDWN	H→L	H	X	X	X	X	X	X
H = High L = Low V = Valid X = Don't care									

Many commands require a delay before the next command may be issued; sometimes the delay depends on the type of the next command. These delay requirements are managed by the values programmed in the memory controller configuration registers (SDCFG1, SDCFG2).

### 17.4.1.1 Activate Command (ACTV)

The ACTV command is responsible for latching the row and bank address and activating the specified row in the memory array. Once the row is activated it can be accessed using subsequent READ and WRITE commands.

#### NOTE

The SDRAMC will support one active row for each chip select block.  
See Section 17.6.1, “Page Management” for more information.

### 17.4.1.2 Read Command (READ)

When the SDRAMC receives a read request, it first checks the row and bank of the new access. If the address falls within the active row of an active bank, it is a page hit, and the READ is issued as soon as possible (pending any delays required by previous commands). If the address is within the active row, but the needed bank is inactive, or if there is no active row, the memory controller will issue an ACTV followed by the READ command. If the address is not within the active row, the memory controller will issue a PALL command to close the active row. Then the SDRAMC issues ACTV to activate the necessary bank and row for the new access, followed finally by the READ to the SDRAM.

The PALL and ACTV commands (if necessary) can sometimes be issued in parallel with an on-going data movement.

All reads, whether burst or single, must be allowed to complete the entire burst length on the memory bus. With DDR memory, the data masks are *asserted* throughout the entire read burst length; but DDR memory ignores the data masks during reads.

### 17.4.1.3 Write Command (WRITE)

When the memory controller receives a write request, it first checks the row and bank of the new access. If the address falls within the active row of an active bank, it is a page hit, and the WRITE is issued as soon as possible (pending any delays required by previous commands). If the address is within the active row but the needed bank is inactive, or if there is no active row, the memory controller will issue an ACTV followed by the WRITE command. If the address is not within the active row, the memory controller will issue a PALL command to close the active row. Then the SDRAMC issues ACTV to activate the necessary row and bank for the new access, followed finally by the WRITE command.

The PALL and ACTV commands (if necessary) can sometimes be issued in parallel with an on-going data movement.

With DDR memory, a read command can be issued overlapping the masked beats at the end of a previous single write

### 17.4.1.4 Precharge All Banks Command (PALL)

The precharge command puts SDRAM into an idle state. The SDRAM must be in this idle state before a REF, LMR, LEMR, or ACTV command to open a new row within a particular bank can be issued.

The memory controller issues the PALL command only when necessary for one of the following conditions:

- Access to a new row
- Refresh interval elapsed
- Software commanded precharge

#### NOTE

The SDRAMC does not support the precharge selected bank memory command.

### 17.4.1.5 Load Mode/Extended Mode Register Command (LMR, LEMR)

All SDRAM devices contain mode registers that are used to configure the timing and burst mode for the SDRAM. These commands are used to access the mode registers that physically reside within the SDRAM devices. During the LMR or LEMR command the SDRAM will latch the address bus and load the value into the selected mode register.

#### NOTE

The LMR and LEMR commands are only used during SDRAM initialization.

The following steps should be used to write the mode register and extended mode register:

1. Set the SDCR[MODE\_EN] bit.
2. Write the SDMR[BA] bits to select the mode register.
3. Write the desired mode register value to the SDMR[ADDR]. Don't overwrite the SDMR[BA] values.
4. Set the SDMR[CMD] bit.
5. Perform a dummy read/write access to an address within an SDRAM block.
6. Repeat step 2 for the extended mode register.
7. Clear the SDCR[MODE\_EN] bit.

#### 17.4.1.5.1 Mode Register Definition

Figure 17-4 shows the mode register definition. Note that this is the SDRAM's mode register not the SDRAMC's mode/extended mode register (SDMR) defined in Section 17.5.1, "SDRAM Mode/Extended Mode Register (SDMR)."

	BA1	BA0	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
Field	0	0	OP_MODE					CASL		BT	BLEN			

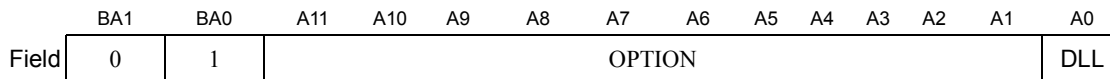
**Figure 17-4. Mode Register**

**Table 17-4. Mode Register Field Descriptions**

Address Line	Description
BA[1:0]	Bank Address. These must both be zero to select the mode register.
A11–A7	Operating Mode. 00000 Normal Operation 00010 Reset DLL Other values should not be used.
A6–A4	CAS latency. Delay in clocks from issuing a READ to valid data out. Check the SDRAM manufacturer's spec as the CASL settings supported can vary from memory to memory.
A3	Burst Type. 0 Sequential 1 Interleaved. This setting should not be used since the SDRAMC does not support interleaved bursts.
A2–A0	Burst length. Determines the number of locations that are accessed for a single READ or WRITE. 111 8 (This is the only valid setting for the MCF5275.) All other values reserved.

### 17.4.1.5.2 Extended Mode Register Definition

Figure 17-5 shows the extended mode register used by DDR SDRAMs. Note that this is the SDRAM's extended mode register not the SDRAMC's mode/extended mode register (SDMR) defined in Section 17.5.1, "SDRAM Mode/Extended Mode Register (SDMR)."



**Figure 17-5. Extended Mode Register**

**Table 17-5. Extended Mode Register Field Descriptions**

Address Line	Description
BA[1:0]	Bank Address. 00 Does not select the extended mode register 01 Selects the extended mode register 1x Reserved
A11–A1	Option. These bits are not defined by the DDR specification. Each DDR SDRAM manufacturer can use these bits to implement optional features. Check with SDRAM manufacturer to determine if any optional features have been implemented. For normal operation all bits should be cleared.
A0	Delay locked loop. Controls enabling of the delay locked loop circuitry used for DDR timing. 0 Enabled 1 Disabled.

### 17.4.1.6 Auto Refresh Command (REF)

The memory controller issues auto refresh commands according to the SDCR[RC] value. Each time the programmed refresh interval elapses, the memory controller issues a PALL command followed by a REF command.

If a memory access is in progress at the time the refresh interval elapses, the memory controller schedules the refresh after the transfer is finished; but the interval timer continues counting so that the average refresh rate is constant.

After REF, the SDRAM is in an idle state and waits for an ACTV command.

### 17.4.1.7 Self Refresh (SREF) and Power Down (PDWN) Commands

The memory controller issues either a PDWN or a SREF command if the SDCR[CKE] bit changes from set to cleared. If the SDCR[REF] bit is set when CKE is cleared, the controller issues a SREF command; if the REF bit is cleared, the controller issues a PDWN command. The REF bit may be changed in the same register write that changes the CKE bit; the controller will act upon the new value of the REF bit.

Just like a REF, the controller automatically issues a PALL command before the self refresh command.

The memory is reactivated from power down or self refresh mode by setting the CKE bit.

If a normal refresh interval elapses while the memory is in self refresh mode, a PALL and REF will be performed as soon as the memory is reactivated. If the memory is put into and brought out of self refresh all within a single refresh interval, the next automatic refresh will occur on schedule.

In self refresh mode, the memory does not require an external clock. To restart periodic refresh when the memory is reactivated, the REF bit must be reasserted; this can be done before the memory is reactivated, or in the same control register write that sets CKE to exit self refresh mode.

## 17.4.2 Power-Up Initialization

SDRAMs have a prescribed initialization sequence. The following sections detail the memory initialization steps for DDR SDRAM. The sequence might change slightly from device-to-device. Refer to the device datasheet as the most relevant reference.

1. After reset is deactivated, pause for the amount of time indicated in the SDRAM specification. Usually 100 $\mu$ s or 200 $\mu$ s.
2. Configure the pin multiplex control for the shared  $\overline{\text{SD\_CS}}$  pins.
3. Write the Base Address and Mask registers (SDBAR0, SDBAR1, SDMR0, and SDMR1) to setup the address space for each chip-select
4. Program the SDRAM configuration registers (SDCFG1 and SDCFG2) with the correct delay and timing values.
5. Issue a PALL command. Initialize the SDRAM control register (SDCR) with SDCR[IPALL] set. The SDCR[REF and IREF] bits should remain cleared for this step. Perform a single read/write to any of the SDRAM chip select address spaces to issue the command.
6. Initialize the SDRAM's extended mode register to enable the DLL. See Section 17.4.1.5, "Load Mode/Extended Mode Register Command (LMR, LEMR)" for instructions on issuing an LEMR command.
7. Initialize the SDRAM's mode register and reset the DLL using the LMR command. See Section 17.4.1.5, "Load Mode/Extended Mode Register Command (LMR, LEMR)" for more instructions on issuing an LMR command. During this step the OP\_MODE field of the mode register should be set to "normal operation/reset DLL."
8. Pause for the DLL lock time specified by the memory.
9. Issue a second PALL command. Initialize the SDRAM control register (SDCR) with SDCR[IPALL] set. The SDCR[REF and IREF] bits should remain cleared for this step. Perform a single read/write to any of the SDRAM chip select address spaces to issue the command.

10. Refresh the SDRAM. The SDRAM spec should indicate a number of refresh cycles to be performed before issuing an LMR command. Write to the SDCR with the IREF bit set (SDCR[REF and IPALL] should be cleared). This will force a refresh of the SDRAM each time the IREF bit is set. Repeat this step until the specified number of refresh cycles have been completed.
11. Initialize the SDRAM's mode register using the LMR command. See Section 17.4.1.5, "Load Mode/Extended Mode Register Command (LMR, LEMR)" for more instruction on issuing an LMR command. During this step the OP\_MODE field of the mode register should be set to "normal operation."
12. Set SDCR[REF] to enable automatic refreshing, and clear SDCR[MODE\_EN] to lock the SDMR. SDCR[IREF and IPALL] remain cleared.

## 17.5 Memory Map/Register Definition

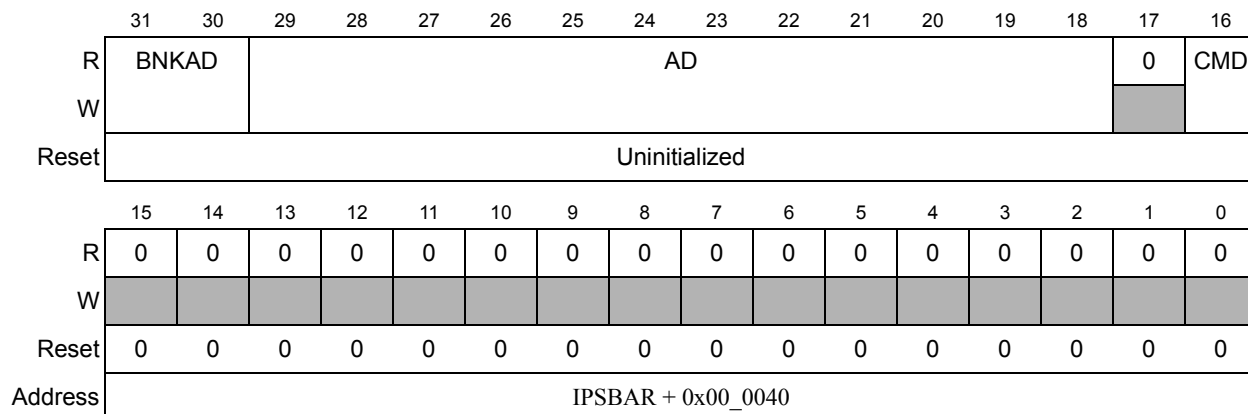
The SDRAM controller contains eight programming registers.

**Table 17-6. SDRAMC Memory Map**

IPSBAR Offset	[31:24]	[23:16]	[15:8]	[7:0]	Access
0x00_0040	SDRAM Mode/Extended Mode Register (SDMR)				R/W
0x00_0044	SDRAM Control Register (SDCR)				R/W
0x00_0048	SDRAM Configuration Register 1 (SDCFG1)				R/W
0x00_004C	SDRAM Configuration Register 2 (SDCFG2)				R/W
0x00_0050	SDRAM Base Address Register 0 (SDBAR0)				R/W
0x00_0054	SDRAM Address Mask Register 0 (SDMR0)				R/W
0x00_0058	SDRAM Base Address Register 1 (SDBAR1)				R/W
0x00_005C	SDRAM Address Mask Register 1 (SDMR1)				R/W

### 17.5.1 SDRAM Mode/Extended Mode Register (SDMR)

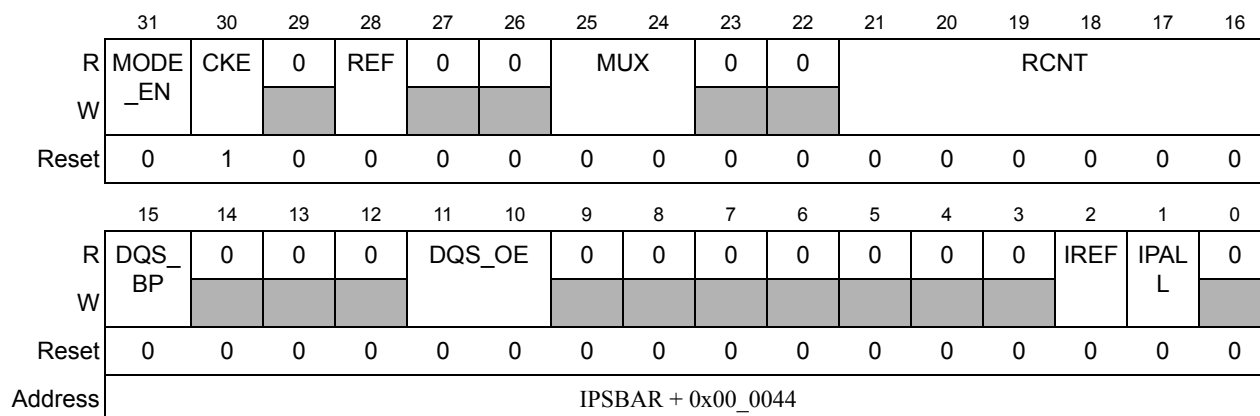
The SDMR is used to write to the mode and extended mode registers that physically reside within in the SDRAM chips. These registers must be programmed during SDRAM initialization. See Section 17.4.2, "Power-Up Initialization" for more information on the initialization sequence.

**Figure 17-6. SDRAM Mode/Extended Mode Register (SDMR)****Table 17-7. SDMR Field Descriptions**

Bits	Name	Description
31–30	BNKAD	Bank address. Driven onto SD_BA[1:0] along with a LMR/LEMR command. All SDRAM chip selects are asserted simultaneously. SDCR[CKE] must be set before attempting to generate an LMR/LEMR command. The SD_BA[1:0] value is used to select between LMR and LEMR commands. 00 Load mode register command (LMR) 01 Load extended mode register command (LEMR) 10–11 Reserved
29–18	AD	Address. Driven onto A[11:0] along with an LMR/LEMR command. The AD value is stored as the mode (or extended mode) register data.
17	—	Reserved. Should be cleared.
16	CMD	Command. 0 Do not generate any command 1 Generate an LMR/LEMR command
15–0	—	Reserved

## 17.5.2 SDRAM Control Register (SDCR)

The SDCR controls SDRAMC operating modes including the refresh count and address line muxing.

**Figure 17-7. SDRAM Control Register (SDCR)****Table 17-8. SDCR Field Descriptions**

Bits	Name	Description																													
31	MODE_EN	Mode enable. 0 Mode register locked, cannot be written 1 Mode register enabled, can be written																													
30	CKE	Clock enable. 0 SDCKE is negated (low) 1 SDCKE is asserted (high)																													
29	—	Reserved, should be cleared.																													
28	REF	Refresh enable. 0 Automatic refresh disabled 1 Automatic refresh enabled																													
27–26	—	Reserved, should be cleared.																													
25–24	MUX	Muxing control. Selects routing of the internal addr[26:23] signals as row or column address bits as shown below and in Table 17-2. <div><table><tr><th rowspan="2">MUX</th><th colspan="4">Internal address bits</th></tr><tr><th>26</th><th>25</th><th>24</th><th>23</th></tr><tr><td>00</td><td>CA12</td><td>CA11</td><td>CA9</td><td>CA8</td></tr><tr><td>01</td><td>CA11</td><td>CA9</td><td>CA8</td><td>RA12</td></tr><tr><td>10</td><td>CA9</td><td>CA8</td><td>RA13</td><td>RA12</td></tr><tr><td>11</td><td colspan="4">Reserved</td></tr></table></div>	MUX	Internal address bits				26	25	24	23	00	CA12	CA11	CA9	CA8	01	CA11	CA9	CA8	RA12	10	CA9	CA8	RA13	RA12	11	Reserved			
MUX	Internal address bits																														
	26	25	24	23																											
00	CA12	CA11	CA9	CA8																											
01	CA11	CA9	CA8	RA12																											
10	CA9	CA8	RA13	RA12																											
11	Reserved																														
23–22	—	Reserved, should be cleared.																													



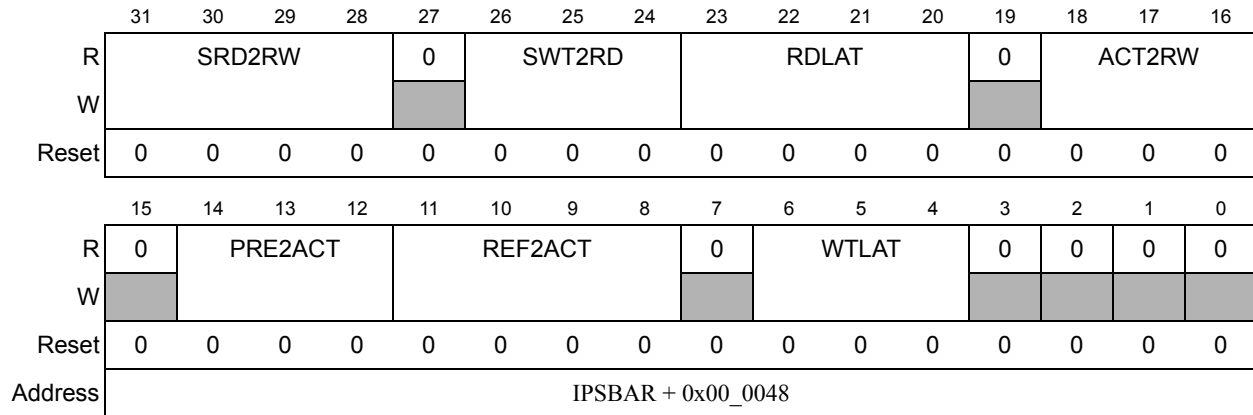
Table 17-8. SDCR Field Descriptions (Continued)

Bits	Name	Description
21–16	RCNT	Automatic Refresh Count. Controls automatic refresh frequency. The number of bus clocks between refresh cycles is $(RC + 1) \times 64$ . $RCNT = (t_{REF} \times CLKOUT / 64) - 1$ , rounded down to the next integer value. Where $t_{REF}$ is the desired refresh period obtained from the SDRAM device data sheet.  Example: $t_{REF}=5\mu s$ , $CLKOUT=75MHz$ . $RCNT = \text{floor}(5 \times 83 / 64) - 1 = 0x04$
15	DQS_BP	DQS Clock Recovery Bypass. Bypasses circuitry for read clock recovery from DQS lines. When bypass is enabled, read clock is generated internally rather than being recovered from DQS pins. 0 Bypass disabled. Read clock recovered from DQS pins 1 Bypass enabled. Read clock generated internally
14–12	—	Reserved. Should be cleared.
11–10	DQS_OE	$\overline{SD\_DQS}$ output enable. 00 Disable all $\overline{SD\_DQS}$ outputs 01 Enable only $\overline{SD\_DQS2}$ 10 Enable only $\overline{SD\_DQS3}$ . 11 Enable both $\overline{SD\_DQS}$ pins <b>Note:</b> For 01 and 10 settings, some 32-bit DDR devices only have a single $\overline{SD\_DQS}$ pin. Enable one of the signals and disable the other. Then short both pins external to the device.
9–3	—	Reserved, should be cleared.
2	IREF	Initiate Auto-Refresh (REF) command. Used to force a software initiated Auto-Refresh command. 0 Do not generate an REF command. 1 An REF command is issued when the next single read/write transaction is performed in the memory controller. All $\overline{SD\_CSn}$ signals are asserted simultaneously. SDCR[CKE] and SDCR[MODE_EN] must be set before attempting to generate an REF command.
1	IPALL	Initiate Precharge All (PALL) command. Used to force a software initiated PALL command. 0 Do not generate a PALL command. 1 A PALL command is issued when the next single read/write transaction is performed to the memory controller. All $\overline{SD\_CSn}$ signals are asserted simultaneously. SDCR[CKE] and SDCR[MODE_EN] must be set before attempting to generate a PALL command.
0	—	Reserved, should be cleared.

### 17.5.3 SDRAM Configuration Register 1 (SDCFG1)

The 32-bit read/write SDRAM configuration register 1 (SDCFG1) stores delay values necessary between specific SDRAM commands. During initialization, software loads values to the register according to the selected CLKOUT frequency and SDRAM specifications. This register is reset only by a power-up reset signal.

The read and write latency fields govern the relative timing of commands and data, and must be exact values. All other fields govern the relative timing from one command to another, they have minimum values but any larger value is also legal (but with decreased performance).

**Figure 17-8. SDRAM Configuration Register 1 (SDCFG1)****Table 17-9. SDCFG1 Field Descriptions**

Bits	Name	Description
31–28	SRD2RW	Single Read to Read/Write/Precharge delay. Limiting case is usually Write to Precharge.  $SRD2RW = \text{Burst Length} / 2$ <b>Note:</b> This formula provides the most optimum (minimum) value. A larger value is also valid, but will reduce performance. Smaller values are reserved.
27	—	Reserved, should be cleared
26–24	SWT2RD	Single Write to Read/Write/Precharge delay. Limiting case is Write to Precharge.  $SWT2RD = (t_{WR} / t_{DDR\_CLKOUT}) + 1$ (Round up to nearest integer) <b>Note:</b> This formula provides the most optimum (minimum) value. A larger value is also valid, but will reduce performance. Smaller values are reserved.
23–20	RDLAT	Read CAS Latency. Read latency. Read command to read data available delay counter.  $RDLAT = 2 \times (CASL + 1)$ <b>Note:</b> Only values 0x6 and 0x7 are valid, the remaining are reserved.
19	—	Reserved, should be cleared.
18–16	ACT2RW	Active to Read/Write delay. Active command to any following read or write delay counter.  $ACT2RW = (t_{RCD} / t_{DDR\_CLKOUT}) - 1$ (Round up to nearest integer)
15	—	Reserved. Should be cleared.
14–12	PRE2ACT	Precharge to Active delay. Precharge command to following Active command delay counter.  $PRE2ACT = (t_{RP} / t_{DDR\_CLKOUT}) - 1$ (Round up to nearest integer) Example: If $t_{RP} = 20\text{ns}$ and $t_{DDR\_CLKOUT} = 1 / 83\text{MHz}$ $20\text{ns} \times 75\text{MHz} = 1.5$ ; round to 2; write 0x1.
11–8	REF2ACT	Refresh to Active delay. Refresh command to following Active or Refresh command delay counter.  $REF2ACT = (t_{RFC} / t_{DDR\_CLKOUT}) - 1$ (Round up to nearest integer)

**Table 17-9. SDCFG1 Field Descriptions (Continued)**

Bits	Name	Description
7	—	Reserved. Should be cleared.
6–4	WTLAT	Write CAS latency. Write command to write data delay counter.  WTLAT = $t_{DQSS} + 2$ Write 0x3 for $t_{DQSS}=1$ . All other values are reserved.
3–0	—	Reserved. Should be cleared.

### 17.5.4 SDRAM Configuration Register 2 (SDCFG2)

The 32-bit read/write SDRAM configuration register 2 stores delay values necessary between specific SDRAM commands. During initialization, software loads values to the register according to the SDRAM information obtained from the data sheet. This register is reset only by a power-up reset signal.

The burst length (BL) field must be exact. All other fields govern the relative timing from one command to another, they have minimum values but any larger value is also legal (but with decreased performance).

All delays in this register are expressed in CLKOUT.

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	BRD2PRE				BWT2RW				BRD2WT				BL			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1
	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
Address	IPSBAR + 0x00_004C															

**Figure 17-9. SDRAM Configuration Register 2 (SDCFG2)****Table 17-10. SDCFG2 Field Descriptions**

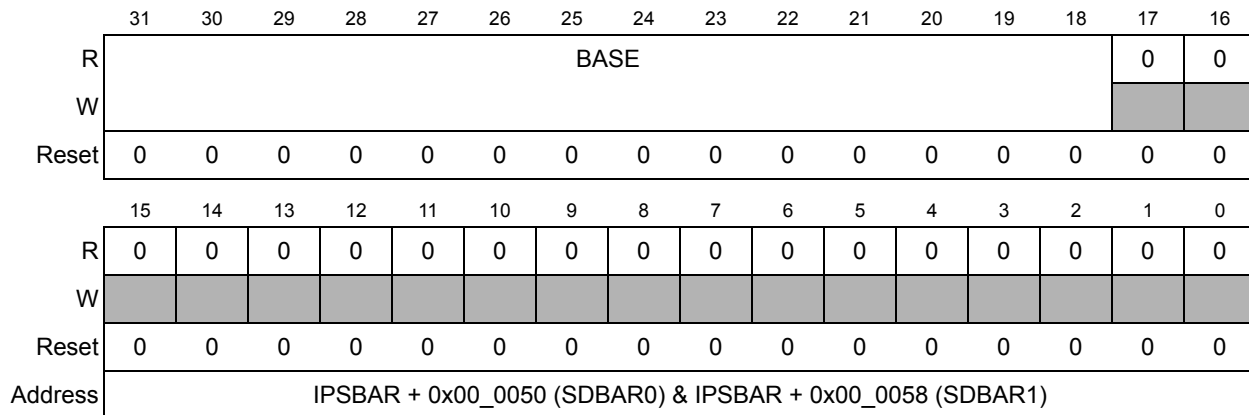
Bits	Name	Description
31–28	BRD2PRE	Burst Read to Read/Precharge delay. Limiting case is Read to Read. BRD2PRE = Burst Length / 2 Write 0x4, all other values are reserved.
27–24	BWT2RW	Burst Write to Read/Write/Precharge delay. Limiting case is Write to Precharge. BWT2RW = (BurstLength / 2 + $t_{WR}$ ) Suggest value = 0x6

**Table 17-10. SDCFG2 Field Descriptions (Continued)**

Bits	Name	Description
23–20	BRD2WT	Burst Read to Write delay. Write 0xB for best performance. All values less than 0xB are reserved.
19–16	BL	Burst Length. BL = Burst Length - 1 Write 0x7, all other values are reserved.
15–0	—	Reserved. Should be cleared.

### 17.5.5 SDRAM Base Address Registers (SDBAR0 & SDBAR1)

The SDRAM Base Address Registers contain the base address for the address spaces associated with  $\overline{SD\_CS0}$  (SDBAR0) and  $\overline{SD\_CS1}$  (SDBAR1). The memory controller compares these values with the current internal address value to decode hits to each CS space. If an address falls within the address space of both chip selects, then  $\overline{SD\_CS0}$  space will take precedence.

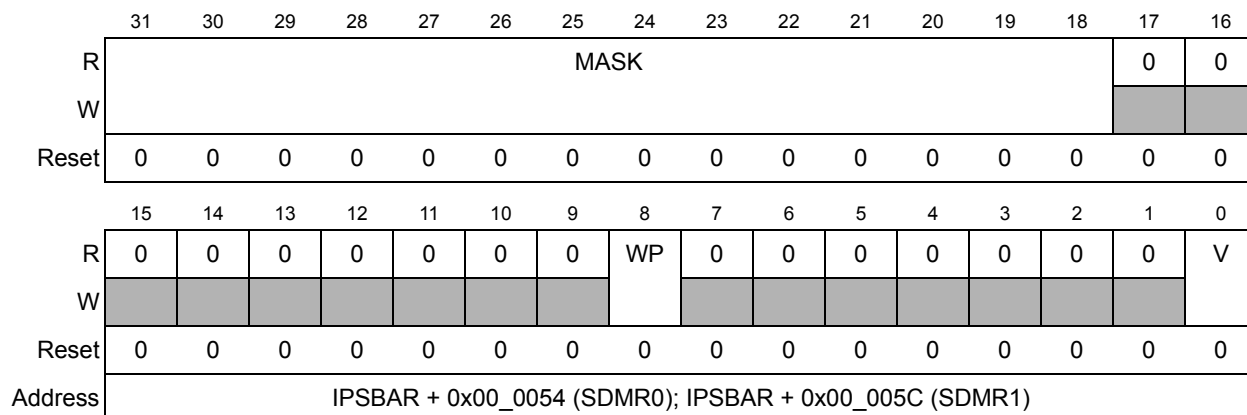
**Figure 17-10. SDRAM Base Address Register (SDBAR<sub>n</sub>)****Table 17-11. SDBAR<sub>n</sub> Field Descriptions**

Bits	Name	Description
31–18	BASE	The Base Address together with $\overline{SDMRn}[\text{MASK}]$ determine the address range in which the associated CS space is located. Each base address bit is compared with the corresponding system address of the current bus cycle. If all unmasked bits match, the address hits in the associated CS space.
17–0	—	Reserved. Should be cleared.

### 17.5.6 SDRAM Address Mask Registers (SDMR0 & SDMR1)

The SDRAM Address Mask Registers mask up to 14 corresponding  $\overline{SDBARn}[\text{BASE}]$  bits. The 18 lsbs of the 32bit internal address do not participate in the address matching in selecting a CS space for access. Masking address bits independently allows external devices of different size

address ranges to be used. Address mask bits can be set or cleared in any order in the field allowing a resource to reside in more than one area of the address map.



**Figure 17-11. SDRAM Address Mask Register (SDMR<sub>n</sub>)**

**Table 17-12. SDMR<sub>n</sub> Field Descriptions**

Bits	Name	Description
31–18	MASK	Base Address Mask masks the associated SDBAR <sub>n</sub> [BASE] bits. Lets the memory controller occupy various address spaces. Mask bits need not be contiguous. 0 The associated address bit is used in decoding a hit to this CS space 1 The associated address bit is not used in the hit decode
17–9	—	Reserved, should be cleared.
8	WP	Write Protect. If a write is performed to this CS space and WP is enabled, the memory controller will assert a transfer error interrupt to the core, without executing the transfer on the external bus. 0 Writes are allowed to this CS space 1 Writes are not allowed to this CS space
7–1	—	Reserved, should be cleared.
0	V	Chip Select Valid enables the CS space. V is cleared at reset to ensure that the CS space is not erroneously decoded. 0 Do not decode hits to this CS space 1 Address hits to this CS space can be decoded

## 17.6 Functional Overview

### 17.6.1 Page Management

SDRAM devices have four internal banks. A particular row and bank of memory must be activated to allow read and write accesses. The SDRAM controller supports paging mode to maximize the memory access throughput. During operation, the SDRAM controller maintains an open page address for each  $\overline{\text{SD\_CS}}$  block. An open page is composed of the active rows in the internal banks.

SDRAMs can have a different row address open in each bank, but the SDRAMC does not support this. The page size of a  $\overline{SD\_CS}$  block is equal to the space size divided by the number of rows; but the page may not be contiguous in the internal address space because the internal address bits used for memory column address [11:8] and column address [7:0] are not consecutive.

Because the column address may be split across two portions of the internal address, the contiguous page size is (number of banks)  $\times$  (256 columns)  $\times$  (number of bits). This gives a contiguous page size of 4 Kbytes. However, the total (possibly fragmented) page size is (number of banks)  $\times$  (number of columns)  $\times$  (number of bits).

If a new access does not fall in the open page of a  $\overline{SD\_CS}$  block, the open page must be closed (PALL) and the new page must be opened (ACTV), then the READ or WRITE command can proceed. An ACTV command only activates one bank of a page. If another read or write falls in an inactive bank of the open page, another ACTV is needed but no precharge is needed. If a read or write falls in any of the active banks of the open page, no PALL or ACTV is needed; the read or write command can be issued immediately.

A page is kept open until one of the following conditions occurs:

- an access outside the open page
- a refresh cycle is started.

All  $\overline{SD\_CS}$  blocks are refreshed at the same time; the refresh closes all banks of every SDRAM block.

## 17.6.2 Transfer Size

In the MCF5275, the internal data bus is 32 bits wide, while the SDRAM external interface bus is 16 bits wide. Therefore, each internal bus data beat requires two memory data beats. The SDRAM controller manages the size translation (packing/unpacking) between 32- and 16-bit buses.

The SDRAM controller supports all possible internal bus transfer sizes. SDRAMs are “burst only” devices; unnecessary beats on the memory bus are masked (write) or discarded (read).

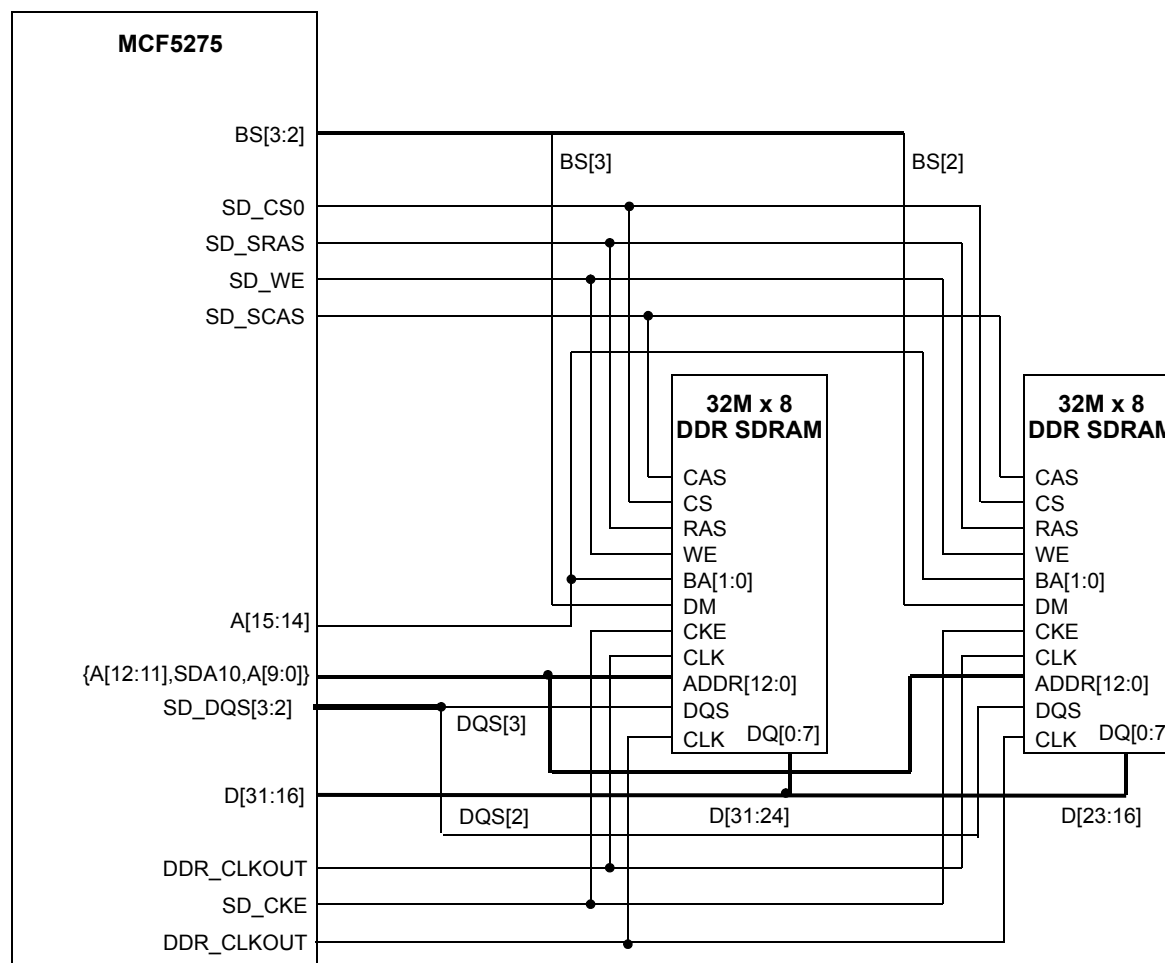
The SDRAMC will perform line bursts (16 byte) for all SDRAM access. This requires eight beats of 2 bytes (one word) on the memory bus. The SDRAM controller transfers the critical longword first, followed by the next three sequential longwords.

The burst size and transfer order must be programmed in the SDRAM mode registers during initialization (SDMR); the burst size also must be programmed in the memory controller (SDCFG2).

In a write operation, the data masks/byte selects, BS[3:2], are used to inhibit writing unused bytes of each beat. In a read operation, the excess read data is discarded.

## 17.7 DDR SDRAM Example

This example interfaces two 32M × 8-bit DDR SDRAM components to an MCF5275 operating at a 83 MHz DDR\_CLKOUT frequency with a single chip select. Figure 17-12 shows a block diagram of the connections and Table 17-13 lists design specifications for this example.



**Figure 17-12. MCF5275 to 64-Mbyte DDR SDRAM Connection**

The devices are organized as 4 internal banks with 13 row address lines 10 column address lines

**Table 17-13. SDRAM Example Specifications**

Parameter	Specification
13 row and 10 column addresses	
Two bank-select lines to access four internal banks	
Allowable burst lengths	2, 4, or 8
CAS latency	From SDRAM data sheet

**Table 17-13. SDRAM Example Specifications (Continued)**

Parameter	Specification
Clock cycle time ( $t_{CK}$ )	From SDRAM data sheet
ACTV-to-read/write delay ( $t_{RCD}$ )	From SDRAM data sheet
Write recovery timer ( $t_{WR}$ )	From SDRAM data sheet
Precharge command to ACTV command ( $t_{RP}$ )	From SDRAM data sheet
Auto refresh command period ( $t_{RFC}$ )	From SDRAM data sheet
Average periodic refresh interval ( $t_{REFI}$ )	From SDRAM data sheet

### 17.7.1 SDRAM Chip Select Settings

For this example, the SDRAM will be connected to  $\overline{SD\_CS0}$  with a base address of 0x0. All other chip selects are unused and do not need to be initialized. The SDBAR0 should be programmed as shown in Figure 17-13.

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Field	BASE															—
Setting	0000_0000_0000_0000															
(hex)	0				0				0				0			

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Field	—															
Setting	0000_0000_0000_0000															
(hex)	0				0				0				0			

**Figure 17-13. SDRAM Chip Select 0 Base Address Register Settings (SDBAR0)**

This configuration results in a value of SDBAR0 = 0x0000\_0000, as described in Table 17-14.

**Table 17-14. SDBAR0 Field Descriptions**

Bits	Name	Setting	Description
31–18	BASE	0	Base address is set to 0x0
17–0	—	0	Reserved. Should be cleared.



	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Field	MASK														—	
Setting	0000_0011_1111_1100															
(hex)	0				3				F				C			

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Field	—							WP	—							V
Setting	0000_0000_0000_0001															
(hex)	0				0				0				1			

**Figure 17-14. SDRAM Chip Select 0 Mask Register Settings (SDMR0)**

This configuration results in a value of SDMR0 = 0x03FC\_0001, as described in Table 17-15.

**Table 17-15. SDMR0 Field Descriptions**

Bits	Name	Setting	Description
31–18	MASK	0x03FC	Address mask for a 64MB space.
17–9	—	0	Reserved. Should be cleared.
8	WP	0	Memory space not write protected.
7–1	—	0	Reserved. Should be cleared.
0	V	1	Memory space is valid.

## 17.7.2 SDRAM Configuration 1 Register Settings

The SDCFG1 register should be programmed as shown in Figure 17-15.

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
Field	SRD2RW				—	SWT2RD				RDLAT				—	ACT2RW		
Setting	0010_0011_0110_0001																
(hex)	2				3				6				1				

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Field	—	PRE2ACT			REF2ACT					WTLAT			—			
Setting	0001_0110_0011_0000															
(hex)	1				6				3				0			

**Figure 17-15. SDRAM Example Configuration Register 1 Settings (SDCFG1)**

This configuration results in a value of SDCFG1 = 0x2361\_1630, as described in Table 17-16.

**Table 17-16. SDCFG1 Field Descriptions**

Bits	Name	Setting	Description
31–28	SRD2RW	0010	$SRD2RW = \text{burst length}/2 = 2$
27	—	0	Reserved. Should be cleared.
26–24	SWT2RD	011	$SWT2RD = t_{WR}/t_{DDR\_CLKOUT} + 1 = 15\text{ns}/12\text{ns} + 1 = 1.25 + 1 = 2.25$ clocks, rounded up to 3
23–20	RDLAT	0110	0x6 is the recommended value for DDR memory with a CASL of 2
19	—	0	Reserved. Should be cleared.
18–16	ACT2RW	001	$ACT2RW = t_{RCD}/t_{DDR\_CLKOUT} - 1 = 18\text{ns}/12\text{ns} - 1 = 1.5 - 1 = 0.5$ , rounded up to 1
15	—	0	Reserved. Should be cleared.
14–12	PRE2ACT	001	$PRE2ACT = t_{RP}/t_{DDR\_CLKOUT} - 1 = 18\text{ns}/12\text{ns} - 1 = 1.5 - 1 = 0.5$ , rounded up to 1
11–8	REF2ACT	0110	$REF2ACT = t_{RFC}/t_{DDR\_CLKOUT} - 1 = 75\text{ns}/12\text{ns} - 1 = 6.25 - 1 = 5.25$ rounded up to 6
7	—	0	Reserved. Should be cleared.
6–4	WTLAT	011	0x3 is the recommended value for DDR
3–0	—	0	Reserved. Should be cleared.

### 17.7.3 SDRAM Configuration 2 Register Settings

The SDCFG2 register should be programmed as shown in Figure 17-16.

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Field	BRD2PRE				BWT2RW				BRD2WT				BL			
Setting	0010_0100_0011_0011															
(hex)	2				4				3				3			
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Field	—															
Setting	0000_0000_0000_0000															
(hex)	0				0				0				0			

**Figure 17-16. SDRAM Example Configuration Register 2 Settings (SDCFG2)**

This configuration results in a value of  $SDCFG2 = 0x2433\_0000$ , as described in Table 17-17.

**Table 17-17. SDCFG2 Field Descriptions**

Bits	Name	Setting	Description
31–28	BRD2PRE	0010	$BRD2PRE = \text{burst length}/2 = 4/2 = 2$
27–24	BWT2RW	0100	$BWT2RW = \text{burst length}/2 + t_{WR} = 4/2 + 2 = 2 + 2 = 4$
23–20	BRD2WT	0011	0x3 is the recommended value for DDR

Table 17-17. SDCFG2 Field Descriptions

Bits	Name	Setting	Description
19–16	BL	0011	BL = burst length - 1 = 4 - 1 = 3
15–0	—	0	Reserved. Should be cleared.

### 17.7.4 SDRAM Control Register Settings and PALL command

The SDCR should be programmed as shown in Figure 17-17. Along with the base settings for the SDCR the MODE\_EN and IPALL bits are set to issue a PALL command to the SDRAM and enable writing of the mode register.

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Field	MODE_EN	CKE	—	REF	—		MUX		—		RCNT					
Setting	1100_0001_0000_1001															
(hex)	C				1				0				9			

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Field	DQS_BP	—			DQS_OE		—						IREF	IPALL	—	
Setting	0000_0000_0000_0010															
(hex)	0				0				0				2			

Figure 17-17. SDRAM Control Register Settings + MODE\_EN and IPALL

This configuration results in a value of SDCR = 0xC109\_0002, as described in Table 17-18. A dummy read/write to SDRAM address space is required to actually send the PALL command.

Table 17-18. SDCR + MODE\_EN and IPALL Field Descriptions

Bits	Name	Setting	Description
31	MODE_EN	1	Mode register is writable.
30	CKE	1	SDCKE is asserted
29	—	0	Reserved. Should be cleared.
28	REF	0	Automatic refresh is disabled
27–26	—	00	Reserved. Should be cleared.
25–24	MUX	01	01 is the MUX setting for a 13 x 10 x 2 memory. See Table 17-2.
23–22	—	0	Reserved. Should be cleared.
21–16	RCNT	001001	$RCNT = (t_{REFI} / (SDCLK \times 64)) - 1 = (7800ns / (12ns \times 64)) - 1 = 9.156$ , round down to 9
15	DQS_BP	0	Bypass disabled. Read clock recovered from DQS pins.
14–12	—	000	Reserved. Should be cleared.

**Table 17-18. SDCR + MODE\_EN and IPALL Field Descriptions (Continued)**

Bits	Name	Setting	Description
11–10	DQS_OE	00	0x0 disables drive for all SDDQS pins for now.
9–3	—	0	Reserved. Should be cleared.
2	IREF	0	Do not initiate a REF command.
1	IPALL	1	Initiate a PALL command.
0	—	0	Reserved. Should be cleared.

### 17.7.5 Set the Extended Mode Register

The SDMR should be programmed as shown in Figure 17-18. This step enables the DDR memory's DLL.

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Field	BNKAD		OPTION											DLL	—	CMD
Setting	0100_0000_0000_0001															
(hex)	4				0				0				1			

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Field	—															
Setting	0000_0000_0000_0000															
(hex)	0				0				0				0			

**Figure 17-18. SDRAM Mode/Extended Mode Register Settings (SDMR)**

This configuration results in a value of SDMR = 0x4001\_0000, as described in Table 17-19. A dummy read/write to SDRAM address space is required to actually send the LEMR command.

**Table 17-19. SDMR Field Descriptions**

Bits	Name	Setting	Description
31–30	BNKAD	01	01 selects the extended mode register.
29–18	OPTION	0	Optional operating modes for the DDR. 0 selects normal operation.
18	DLL	0	Enable the DLL.
17	—	0	Reserved. Should be cleared.
16	CMD	1	Initiate the LEMR command.
15–0	—	0	Reserved. Should be cleared.

## 17.7.6 Set the Mode Register and Reset DLL

The SDMR should be programmed as shown in Figure 17-19. This step programs the mode register and resets the DLL.

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16		
Field	BNKAD		OP_MODE						CASL				BT	BLEN			—	CMD
Setting	0000_0100_1001_1101																	
(hex)	0				4				9				D					

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Field	—															
Setting	0000_0000_0000_0000															
(hex)	0				0				0				0			

**Figure 17-19. SDRAM Mode/Extended Mode Register Settings (SDMR)**

This configuration results in a value of SDMR = 0x049D\_0000, as described in Table 17-20. A dummy read/write to SDRAM address space is required to actually send the LMR command.

**Table 17-20. SDMR Field Descriptions**

Bits	Name	Setting	Description
31–30	BNKAD	00	00 selects the mode register.
29–25	OP_MODE	0010	Selects normal operating mode and resets the DLL.
24–22	CASL	010	CAS latency of two clocks.
21	BT	0	Sequential burst type.
20–18	BLEN	111	Burst length of eight.
17	—	0	Reserved. Should be cleared.
16	CMD	1	Initiate the LMR command.
15–0	—	0	Reserved. Should be cleared.

## 17.7.7 Issue a PALL command

The SDCR should be programmed as shown in Figure 17-20. This will issue a second PALL command to the memory. The same SDCR value calculated in Section 17.7.4, “SDRAM Control Register Settings and PALL command” is used (0xC10D\_0002).

		31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Field		MODE_EN	CKE	—	REF	—		MUX		—		RCNT					
Setting		1100_0001_0000_1001															
(hex)		C				1				0				9			

		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Field		DQS_BP	—			DQS_OE		—						IREF	IPALL	—	
Setting		0000_0000_0000_0010															
(hex)		0				0				0				2			

**Figure 17-20. SDRAM Control Register Settings + MODE\_EN and IPALL**

This configuration results in a value of SDCR = 0xC109\_0002, as described in Table 17-21. A dummy read/write to SDRAM address space is required to actually send the PALL command.

**Table 17-21. SDCR + MODE\_EN and IPALL Field Descriptions**

Bits	Name	Setting	Description
31	MODE_EN	1	Mode register is writable.
30	CKE	1	SDCKE is asserted
29	—	0	Reserved. Should be cleared.
28	REF	0	Automatic refresh is disabled
27–26	—	00	Reserved. Should be cleared.
25–24	MUX	01	01 is the MUX setting for a 13 x 10 x 2 memory. See Table 17-2.
23–22	—	0	Reserved. Should be cleared.
21–16	RCNT	001001	$RCNT = (t_{REFI} / (SDCLK \times 64)) - 1 = (7800ns / (12ns \times 64)) - 1 = 9.156$ , round down to 9
15	DQS_BP	0	Bypass disabled. Read clock recovered from DQS pins.
14–12	—	000	Reserved. Should be cleared.
11–10	DQS_OE	00	0x0 disables drive for all SDDQS pins for now.
9–3	—	0	Reserved. Should be cleared.
2	IREF	0	Do not initiate a REF command.
1	IPALL	1	Initiate a PALL command.
0	—	0	Reserved. Should be cleared.

## 17.8 Perform Two Refresh Cycles

The SDCR should be programmed as shown in Figure 17-21. Along with the base settings for the SDCR the MODE\_EN and IREF bits are set to issue an REF command to the SDRAM and enable

writing of the mode register. The memory used in this example requires two refresh cycles, so this step is repeated twice.

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Field	MODE_EN	CKE	—	REF	—		MUX		—		RCNT					
Setting	1100_0001_0000_1001															
(hex)	C				1				0				9			

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Field	DQS_BP	—			DQS_OE		—						IREF	IPALL	—	
Setting	0000_0000_0000_0100															
(hex)	0				0				0				4			

**Figure 17-21. SDRAM Control Register Settings + MODE\_EN and IREF**

This configuration results in a value of SDCR = 0xC109\_0004, as described in Table 17-22.

**Table 17-22. SDCR + MODE\_EN and IREF Field Descriptions**

Bits	Name	Setting	Description
31	MODE_EN	1	Mode register is writable.
30	CKE	1	SDCKE is asserted
29	—	0	Reserved. Should be cleared.
28	REF	0	Automatic refresh is disabled
27–26	—	00	Reserved. Should be cleared.
25–24	MUX	01	01 is the MUX setting for a 13 x 10 x 2 memory. See Table 17-2.
23–22	—	0	Reserved. Should be cleared.
21–16	RCNT	001001	$RCNT = (t_{REFI} / (SDCLK \times 64)) - 1 = (7800ns / (12ns \times 64)) - 1 = 9.156$ , round down to 9
15	DQS_BP	0	Bypass disabled. Read clock recovered from DQS pins.
14–12	—	000	Reserved. Should be cleared.
11–10	DQS_OE	00	0x0 disables drive for all SDDQS pins for now.
9–3	—	0	Reserved. Should be cleared.
2	IREF	1	Initiate a REF command.
1	IPALL	0	Do not initiate a PALL command.
0	—	0	Reserved. Should be cleared.

## 17.8.1 Clear the Reset DLL Bit in the Mode Register

The SDMR should be programmed as shown in Figure 17-22. This step programs the mode register and enables normal operation of the DLL by clearing the “reset DLL” option.

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
Field	BNKAD		OP_MODE				CASL				BT	BLEN			—	CMD	
Setting	0000_0000_1001_1101																
(hex)	0				0				9				D				

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Field	—															
Setting	0000_0000_0000_0000															
(hex)	0				0				0				0			

**Figure 17-22. SDRAM Mode/Extended Mode Register Settings**

This configuration results in a value of SDMR = 0x009D\_0000, as described in Table 17-23. A dummy read/write to SDRAM address space is required to actually send the LMR command.

**Table 17-23. SDMR Field Descriptions**

Bits	Name	Setting	Description
31–30	BNKAD	00	00 selects the mode register.
29–25	OP_MODE	0000	Selects normal operating mode.
24–22	CASL	010	CAS latency of two clocks.
21	BT	0	Sequential burst type.
20–18	BLEN	111	Burst length of eight.
17	—	0	Reserved. Should be cleared.
16	CMD	1	Initiate the LMR command.
15–0	—	0	Reserved. Should be cleared.

## 17.8.2 Enable Automatic Refresh and Lock Mode Register

The SDCR should be programmed as shown in Figure 17-23. Along with the base settings for the SDCR the REF bit is set to enable automatic refreshing of the memory. In addition, the MODE\_EN bit is cleared to disable write to the SDMR.



		31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Field		MODE_EN	CKE	—	REF	—		MUX		—		RCNT					
Setting		0101_0001_0000_1001															
(hex)		5				1				0				9			

		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Field		DQS_BP	—			DQS_OE		—						IREF	IPALL	—	
Setting		0000_1100_0000_0000															
(hex)		0				C				0				0			

**Figure 17-23. SDRAM Control Register Settings + REF**

This configuration results in a value of SDCR = 0x510D\_0C00, as described in Table 17-24.

**Table 17-24. SDCR + REF Field Descriptions**

Bits	Name	Setting	Description
31	MODE_EN	0	Mode register is not writable.
30	CKE	1	SDCKE is asserted
29	—	0	Reserved. Should be cleared.
28	REF	0	Automatic refresh is disabled
27–26	—	00	Reserved. Should be cleared.
25–24	MUX	01	01 is the MUX setting for a 13 x 10 x 2 memory. See Table 17-2.
23–22	—	0	Reserved. Should be cleared.
21–16	RCNT	001001	$RCNT = (t_{REFI} / (SDCLK \times 64)) - 1 = (7800ns / (12ns \times 64)) - 1 = 9.156$ , round down to 9
15	DQS_BP	0	Bypass disabled. Read clock recovered from DQS pins.
14–12	—	000	Reserved. Should be cleared.
11–10	DQS_OE	11	Enables drive for all $\overline{SD\_DQS}$ pins.
9–3	—	0	Reserved. Should be cleared.
2	IREF	0	Do not initiate a REF command.
1	IPALL	0	Do not initiate a PALL command.
0	—	0	Reserved. Should be cleared.

### 17.8.3 Initialization Code

The following assembly code initializes the DDR SDRAM using the register values determined above.

Basic Configuration and Initialization:

## SDRAM Controller (SDRAMC)

```
move.l #0x00000000, d0 //Initialize SDBAR0
move.l d0, SDBAR0
move.l #0x03FC0001, d0 //Initialize SDMR0
move.l d0, SDMR0
move.l #0x23611630, d0 //Initialize SDCFG1
move.l d0, SDCFG1
move.l #0x24330000, d0 //Initialize SDCFG2
move.l d0, SDCFG2
```

### Precharge Sequence and enable write to SDMR:

```
move.l #0xC1090002, d0 //Initialize SDCR, send PALL, enable SDMR
move.l d0, SDCR

move.l #0x0, a0
move.l #0xdeadbeef, (a0) //dummy write to SDRAM space
```

### Write Extended Mode Register:

```
move.l #0x40010000, d0 //Write LEMR to enable DLL
move.l d0, SDMR

move.l #0x0, a0
move.l #0xdeadbeef, (a0) //dummy write to SDRAM space
```

### Write Mode Register and Reset DLL:

```
move.l #0x049D0000, d0 //Write LMR and reset DLL
move.l d0, SDMR

move.l #0x0, a0
move.l #0xdeadbeef, (a0) //dummy write to SDRAM space
```

### Precharge Sequence:

```
move.l #0xC1090002, d0 //Send PALL
move.l d0, SDCR

move.l #0x0, a0
move.l #0xdeadbeef, (a0) //dummy write to SDRAM space
```

### Refresh Sequence:

```
move.l #0xC1090004, d0 //Send first REF command
move.l d0, SDCR
move.l #0xC1090004, d0 //Send second REF command
move.l d0, SDCR
```

### Write Mode Register and Clear Reset DLL:

```
move.l #0x009D0000, d0 //Write LMR and clear reset DLL
move.l d0, SDMR
```

```
move.l  #0x0,a0  
move.l  #0xdeadbeef,(a0)//dummy write to SDRAM space
```

#### Enable Auto Refresh and Lock SDMR:

```
move.l  #0x51090C00, d0 //Enable auto refresh and clear MODE_EN  
move.l  d0, SDCR
```



# Chapter 18

## DMA Controller Module

### 18.1 Introduction

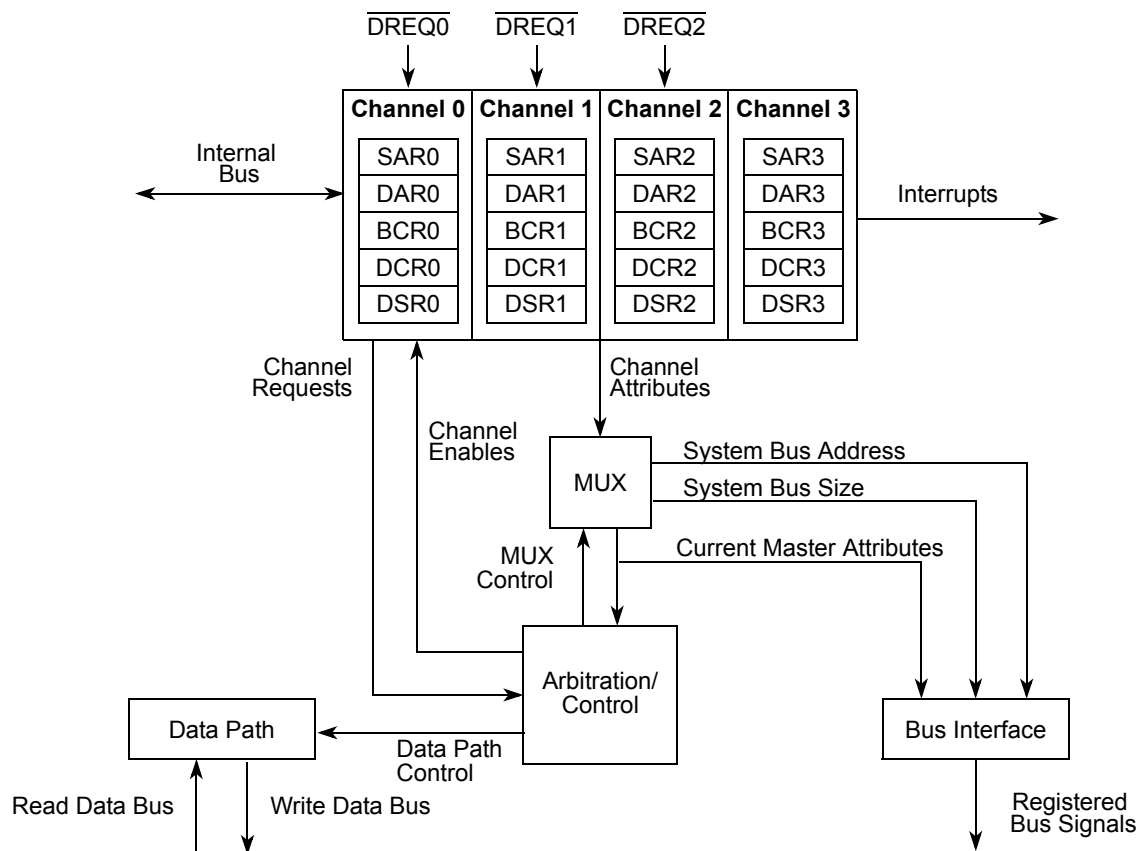
This chapter describes the Direct Memory Access (DMA) controller module. It provides an overview of the module and describes in detail its signals and registers. The latter sections of this chapter describe operations, features, and supported data transfer modes in detail.

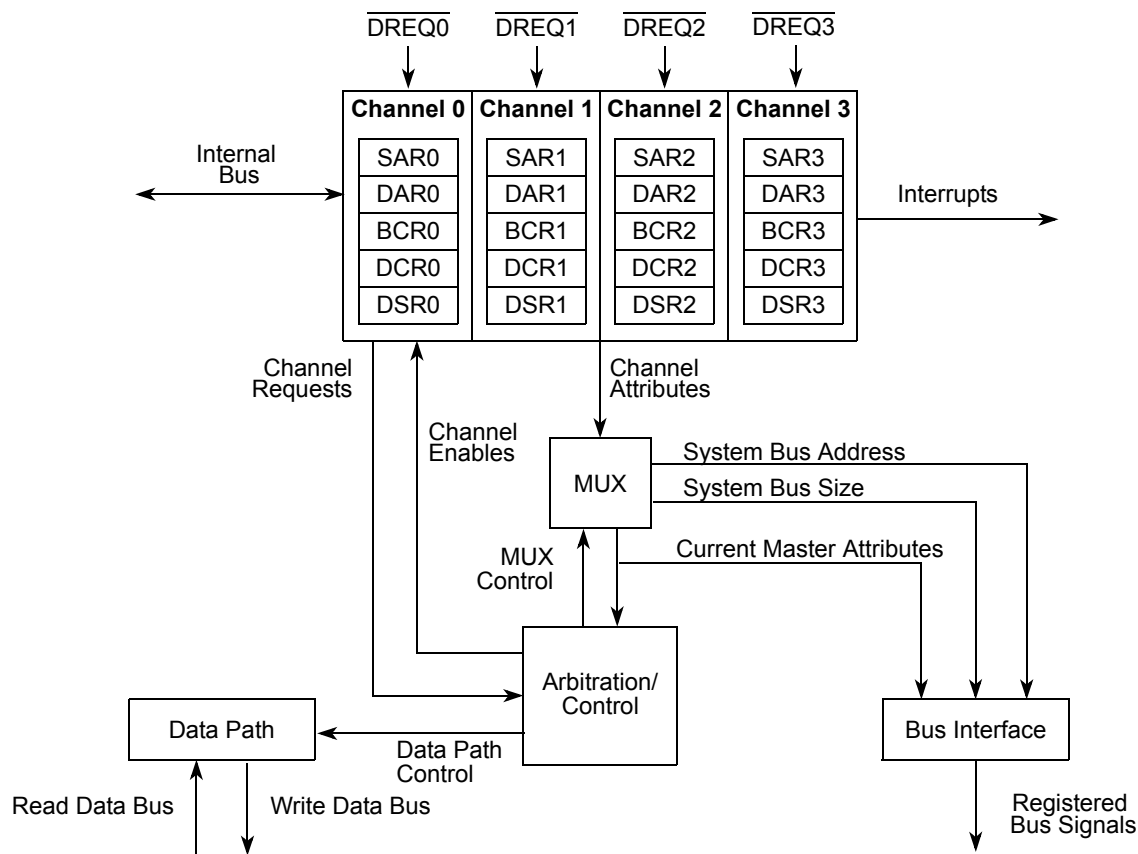
#### NOTE

The designation “*n*” is used throughout this section to refer to registers or signals associated with one of the four identical DMA channels: DMA0, DMA1, DMA2 or DMA3.

#### 18.1.1 Overview

The DMA controller module provides an efficient way to move blocks of data with minimal processor interaction. The DMA module, shown in [Figure 18-1](#), provides four channels that allow byte, word, longword, or 16-byte burst data transfers. Each channel has a dedicated source address register (SAR<sub>*n*</sub>), destination address register (DAR<sub>*n*</sub>), byte count register (BCR<sub>*n*</sub>), control register (DCR<sub>*n*</sub>), and status register (DSR<sub>*n*</sub>). Transfers are dual address to on-chip devices, such as UART and SDRAM controller.





**Figure 18-1. DMA Signal Diagram**

### NOTE

Throughout this chapter “external request” and DREQ are used to refer to a DMA request from one of the on-chip UARTS, DMA timers, or  $\overline{\text{DREQ}}$  signals. For details on the connections associated with DMA request inputs, see [Section 18.3.1, “DMA Request Control \(DMAREQC\).”](#)

## 18.1.2 Features

The DMA controller module features are as follows:

- Four independently programmable DMA controller channels
- Auto-alignment feature for source or destination accesses
- Dual-address transfers
- Channel arbitration on transfer boundaries

- Data transfers in 8-, 16-, 32-, or 128-bit blocks using a 16-byte buffer
- Continuous-mode or cycle-steal transfers
- Independent transfer widths for source and destination
- Independent source and destination address registers
- Modulo addressing on source and destination addresses
- Automatic channel linking

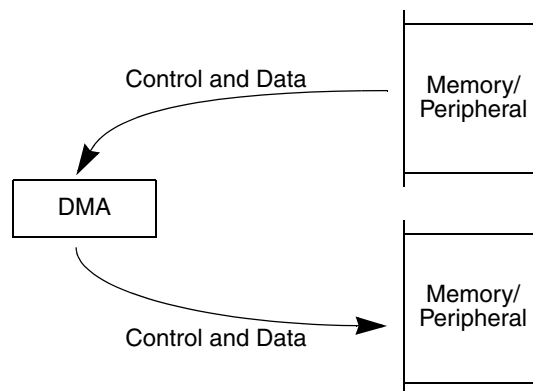
## 18.2 DMA Transfer Overview

The DMA module can move data within system memory (including memory and peripheral devices) with minimal processor intervention, greatly improving overall system performance. The DMA module consists of four independent, functionally equivalent channels, so references to DMA in this chapter apply to any of the channels. It is not possible to implicitly address all four channels at once.

The processor generates DMA requests internally by setting DCR[START]; the UART modules and DMA timers can generate a DMA request by asserting internal DREQ signals. The processor can program bus bandwidth for each channel. The channels support cycle-steal and continuous transfer modes; see [Section 18.4.1, “Transfer Requests \(Cycle-Steal and Continuous Modes\).”](#)

The DMA controller supports dual-address transfers. The DMA channels support up to 32 data bits.

- Dual-address transfers—A dual-address transfer consists of a read followed by a write and is initiated by an internal request using the START bit or by asserting  $\overline{\text{DREQ}}_n$ . Two types of transfer can occur: a read from a source device or a write to a destination device. See [Figure 18-2](#) for more information.



**Figure 18-2. Dual-Address Transfer**



Any operation involving the DMA module follows the same three steps:

1. Channel initialization—Channel registers are loaded with control information, address pointers, and a byte-transfer count.
2. Data transfer—The DMA accepts requests for operand transfers and provides addressing and bus control for the transfers.
3. Channel termination—Occurs after the operation is finished, either successfully or due to an error. The channel indicates the operation status in the channel's DSR, described in [Section 18.3.4.1, “DMA Status Registers \(DSR0–DSR3\).”](#)

## 18.3 Memory Map/Register Definition

This section describes each internal register and its bit assignment. Note that modifying DMA control registers during a DMA transfer can result in undefined operation. [Table 18-1](#) shows the mapping of DMA controller registers.

**Table 18-1. Memory Map for DMA Controller Module Registers**

DMA Channel	IPSBAR Offset	[31:24]	[23:16]	[15:8]	[7:0]
—	0x00_0014	DMA Request Control Register (DMAREQC) <sup>1</sup>			
0	0x00_0100	Source Address Register 0 (SAR0)			
	0x00_0104	Destination Address Register 0 (DAR0)			
	0x00_0108	Status Register 0 (DSR0)	Byte Count Register 0 (BCR0)		
	0x00_010C	Control Register 0 (DCR0)			
1	0x00_0110	Source Address Register 1 (SAR1)			
	0x00_0114	Destination Address Register 1 (DAR1)			
	0x00_0118	Status Register 1 (DSR1)	Byte Count Register 1 (BCR1)		
	0x00_011C	Control Register 1 (DCR1)			
2	0x00_0120	Source Address Register 2 (SAR2)			
	0x00_0124	Destination Address Register 2 (DAR2)			
	0x00_0128	Status Register 2 (DSR2)	Byte Count Register 2 (BCR2)		
	0x00_012C	Control Register 2 (DCR2)			
3	0x00_0130	Source Address Register 3 (SAR3)			
	0x00_0134	Destination Address Register 3 (DAR3)			
	0x00_0138	Status Register 3 (DSR3)	Byte Count Register 3 (BCR3)		
	0x00_013C	Control Register 3 (DCR3)			

<sup>1</sup> Located within the SCM, but listed here for clarity.

## 18.3.1 DMA Request Control (DMAREQC)

The DMAREQC register provides a software-controlled connection matrix for DMA requests. It logically routes DMA requests from the DMA timers and UARTs to the four channels of the DMA controller. Writing to this register determines the exact routing of the DMA request to the four channels of the DMA modules. If  $\overline{DCRn[EEXT]}$  is set and the channel is idle, the assertion of the appropriate  $\overline{DREQn}$  activates channel  $n$ .

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	0	0	0	0	DMAREQC_EXT			
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	DMAC3				DMAC2				DMAC1				DMAC0			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Address	IPSBAR + 0x00_0014															

**Figure 18-3. DMA Request Control Register (DMAREQC)**

**Table 18-2. DMAREQC Field Description**

Bits	Name	Description															
31–20	—	Reserved, should be cleared.															
19-16	DMAREQC_EXT	<p>DMA request control for external (off-chip) requests. The DMAREQC_EXT[3:0] bits correspond to DMA channels 3, 2, 1, and 0. If set, the corresponding DMAC<math>n</math> bit field is ignored. If cleared, refer to the appropriate DMAC<math>n</math> bit field for configuring the internal DMA requestor.</p> <table><tr><td></td><td>DMAREQC_EXT[3]</td><td>DMAREQC_EXT[2]</td><td>DMAREQC_EXT[1]</td><td>DMAREQC_EXT[0]</td></tr><tr><td>0</td><td>See DMAC3</td><td>See DMAC2</td><td>See DMAC1</td><td>See DMAC0</td></tr><tr><td>1</td><td>External <math>\overline{\text{DREQ3}}</math></td><td>External <math>\overline{\text{DREQ2}}</math></td><td>External <math>\overline{\text{DREQ1}}</math></td><td>External <math>\overline{\text{DREQ0}}</math></td></tr></table> <p><b>Note:</b> GPIO must be configured to enable external DMA requests.</p>		DMAREQC_EXT[3]	DMAREQC_EXT[2]	DMAREQC_EXT[1]	DMAREQC_EXT[0]	0	See DMAC3	See DMAC2	See DMAC1	See DMAC0	1	External $\overline{\text{DREQ3}}$	External $\overline{\text{DREQ2}}$	External $\overline{\text{DREQ1}}$	External $\overline{\text{DREQ0}}$
	DMAREQC_EXT[3]	DMAREQC_EXT[2]	DMAREQC_EXT[1]	DMAREQC_EXT[0]													
0	See DMAC3	See DMAC2	See DMAC1	See DMAC0													
1	External $\overline{\text{DREQ3}}$	External $\overline{\text{DREQ2}}$	External $\overline{\text{DREQ1}}$	External $\overline{\text{DREQ0}}$													

**Table 18-2. DMAREQC Field Description (Continued)**

Bits	Name	Description
15–0	DMAC $n$	<p>DMA channel <math>n</math>. Each four bit field defines the logical connection between the DMA requestors and that DMA channel. There are ten possible requestors (4 DMA Timers and 6 UARTs). Any request can be routed to any of the DMA channels. Effectively, the DMAREQC provides a software-controlled routing matrix of the 10 DMA request signals to the 4 channels of the DMA module. DMAC3 controls DMA channel 3, DMAC2 controls DMA channel 2, etc.</p> <p>0100 DMA Timer 0.  0101 DMA Timer 1.  0110 DMA Timer 2.  0111 DMA Timer 3.  1000 UART0 Receive.  1001 UART1 Receive.  1010 UART2 Receive.  1100 UART0 Transmit.  1101 UART1 Transmit.  1110 UART2 Transmit.</p> <p>All other values are reserved and will not generate a DMA request.</p>

### 18.3.2 Source Address Registers (SAR0–SAR3)

SAR $n$ , shown in [Figure 18-4](#), contains the address from which the DMA controller requests data.

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	SAR															
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	SAR															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Address	IPSBAR + 0x00_0100 (DMA0); IPSBAR + 0x00_0110 (DMA1); IPSBAR + 0x00_0120 (DMA2); IPSBAR + 0x00_0130 (DMA3)															

**Figure 18-4. Source Address Registers (SAR $n$ )**

#### NOTE

The backdoor enable bit must be set in the SCM RAMBAR as well as the secondary port valid bit in the Core RAMBAR in order to enable backdoor accesses from the DMA to SRAM. See [Section 11.2.1.2, “Memory Base Address Register \(RAMBAR\)”](#) and [Section 6.2.1, “SRAM Base Address Register \(RAMBAR\)”](#) for more details.

### 18.3.3 Destination Address Registers (DAR0–DAR3)

DAR<sub>*n*</sub>, shown in Figure 18-5, holds the address to which the DMA controller sends data.

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	DAR															
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	DAR															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Address	IPSBAR + 0x00_0104 (DMA0); IPSBAR + 0x00_0114 (DMA1); IPSBAR + 0x00_0124 (DMA2); IPSBAR + 0x00_0134 (DMA3)															

**Figure 18-5. Destination Address Registers (DAR<sub>*n*</sub>)**

#### NOTE

The DMA does not maintain coherency with the cache. Therefore, DMAs should not transfer data to cacheable memory unless software is used to maintain the cache coherency.

### 18.3.4 Byte Count Registers (BCR0–BCR3) and DMA Status Registers (DSR0–DSR3)

BCR<sub>*n*</sub>, shown in Figure 18-6, contains the number of bytes yet to be transferred for a given block. BCR<sub>*n*</sub> decrements on the successful completion of the address transfer of a write transfer. BCR<sub>*n*</sub> decrements by 1, 2, 4, or 16 for byte, word, longword, or line accesses, respectively.

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	DSR								BCR							
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	BCR															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Address	IPSBAR + 0x00_0108 (DMA0); IPSBAR + 0x00_0118 (DMA1); IPSBAR + 0x00_0128 (DMA2); IPSBAR + 0x00_0138 (DMA3)															

**Figure 18-6. Byte Count Registers (BCR<sub>*n*</sub>) and Status Registers (DSR<sub>*n*</sub>)**

DSR $_n$ [DONE], shown in Figure 18-7, is set when the block transfer is complete.

When a transfer sequence is initiated and BCR $_n$ [BCR] is not a multiple of 16, 4, or 2 when the DMA is configured for line, longword, or word transfers, respectively, DSR $_n$ [CE] is set and no transfer occurs. See Section 18.3.4.1, “DMA Status Registers (DSR0–DSR3).”

### 18.3.4.1 DMA Status Registers (DSR0–DSR3)

In response to an event, the DMA controller writes to the appropriate DSR $_n$  bit, Figure 18-7. Only a write to DSR $_n$ [DONE] results in action.

	7	6	5	4	3	2	1	0
R	0	CE	BES	BED	0	REQ	BSY	DONE
W								
Reset	0	0	0	0	0	0	0	0
Address	See Figure 18-6							

**Figure 18-7. DMA Status Registers (DSR $_n$ )**

**Table 18-3. DSR $_n$  Field Descriptions**

Bits	Name	Description
7	—	Reserved, should be cleared.
6	CE	Configuration error. Occurs when BCR, SAR, or DAR does not match the requested transfer size, or if BCR = 0 when the DMA receives a start condition. CE is cleared at hardware reset or by writing a 1 to DSR[DONE]. 0 No configuration error exists. 1 A configuration error has occurred.
5	BES	Bus error on source 0 No bus error occurred. 1 The DMA channel terminated with a bus error during the read portion of a transfer.
4	BED	Bus error on destination 0 No bus error occurred. 1 The DMA channel terminated with a bus error during the write portion of a transfer.
3	—	Reserved, should be cleared.
2	REQ	Request 0 No request is pending or the channel is currently active. Cleared when the channel is selected. 1 The DMA channel has a transfer remaining and the channel is not selected.

**Table 18-3. DSR<sub>n</sub> Field Descriptions (Continued)**

Bits	Name	Description
1	BSY	Busy 0 DMA channel is inactive. Cleared when the DMA has finished the last transaction. 1 BSY is set the first time the channel is enabled after a transfer is initiated.
0	DONE	Transactions done. Set when all DMA controller transactions complete, as determined by transfer count or error conditions. When BCR reaches zero, DONE is set when the final transfer completes successfully. DONE can also be used to abort a transfer by resetting the status bits. When a transfer completes, software must clear DONE before reprogramming the DMA. 0 Writing or reading a 0 has no effect. 1 DMA transfer completed. Writing a 1 to this bit clears all DMA status bits and can be used in an interrupt handler to clear the DMA interrupt and error bits.

### 18.3.5 DMA Control Registers (DCR0–DCR3)

DCR<sub>n</sub>, shown in [Figure 18-8](#), is used for configuring the DMA controller module.

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	INT	EEXT	CS	AA	BWC			0	0	SINC	SSIZE		DINC	DSIZE		0
W																START
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	SMOD				DMOD				D_REQ	0	LINKCC		LCH1		LCH2	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Address	IPSBAR + 0x00_010C (DMA0); IPSBAR + 0x011C (DMA1); IPSBAR + 0x012C (DMA2); IPSBAR + 0x013C (DMA3)															

**Figure 18-8. DMA Control Registers (DCR<sub>n</sub>)****Table 18-4. DCR<sub>n</sub> Field Descriptions**

Bits	Name	Description
31	INT	Interrupt on completion of transfer. Determines whether an interrupt is generated by completing a transfer or by the occurrence of an error condition. 0 No interrupt is generated. 1 Internal interrupt signal is enabled.
30	EEXT	Enable external request. Care should be taken because a collision can occur between the START bit and DREQ <sub>n</sub> when EEXT = 1. 0 External request is ignored. 1 Enables external request to initiate transfer. The internal request (initiated by setting the START bit) is always enabled.

Table 18-4. DCR<sub>n</sub> Field Descriptions (Continued)

Bits	Name	Description																		
29	CS	Cycle steal. 0 DMA continuously makes read/write transfers until the BCR decrements to 0. 1 Forces a single read/write transfer per request. The request may be internal by setting the START bit, or external by asserting DREQ <sub>n</sub> .																		
28	AA	Auto-align. AA and SIZE determine whether the source or destination is auto-aligned, that is, transfers are optimized based on the address and size. See <a href="#">Section 18.4.4.2, “Auto-Alignment.”</a> 0 Auto-align disabled 1 If SSIZE indicates a transfer no smaller than DSIZE, source accesses are auto-aligned; otherwise, destination accesses are auto-aligned. Source alignment takes precedence over destination alignment. If auto-alignment is enabled, the appropriate address register increments, regardless of DINC or SINC.																		
27–25	BWC	Bandwidth control. Indicates the number of bytes in a block transfer. When the byte count reaches a multiple of the BWC value, the DMA releases the bus. <table><tr><th>BWC</th><th>Number of kilobytes per block</th></tr><tr><td>000</td><td>DMA has priority and does not negate its request until transfer completes.</td></tr><tr><td>001</td><td>16 Kbytes</td></tr><tr><td>010</td><td>32 Kbytes</td></tr><tr><td>011</td><td>64 Kbytes</td></tr><tr><td>100</td><td>128 Kbytes</td></tr><tr><td>101</td><td>256 Kbytes</td></tr><tr><td>110</td><td>512 Kbytes</td></tr><tr><td>111</td><td>1024 Kbytes</td></tr></table>	BWC	Number of kilobytes per block	000	DMA has priority and does not negate its request until transfer completes.	001	16 Kbytes	010	32 Kbytes	011	64 Kbytes	100	128 Kbytes	101	256 Kbytes	110	512 Kbytes	111	1024 Kbytes
BWC	Number of kilobytes per block																			
000	DMA has priority and does not negate its request until transfer completes.																			
001	16 Kbytes																			
010	32 Kbytes																			
011	64 Kbytes																			
100	128 Kbytes																			
101	256 Kbytes																			
110	512 Kbytes																			
111	1024 Kbytes																			
24–23	—	Reserved, should be cleared.																		
22	SINC	Source increment. Controls whether a source address increments after each successful transfer. 0 No change to SAR after a successful transfer. 1 The SAR increments by 1, 2, 4, or 16, as determined by the transfer size.																		
21–20	SSIZE	Source size. Determines the data size of the source bus cycle for the DMA control module. 00 Longword 01 Byte 10 Word 11 Line (16-byte burst)																		
19	DINC	Destination increment. Controls whether a destination address increments after each successful transfer. 0 No change to the DAR after a successful transfer. 1 The DAR increments by 1, 2, 4, or 16, depending upon the size of the transfer.																		

**Table 18-4. DCR<sub>n</sub> Field Descriptions (Continued)**

Bits	Name	Description												
18–17	DSIZE	Destination size. Determines the data size of the destination bus cycle for the DMA controller. 00 Longword 01 Byte 10 Word 11 Line (16-byte burst)												
16	START	Start transfer. 0 DMA inactive 1 The DMA begins the transfer in accordance to the values in the control registers. START is cleared automatically after one system clock and is always read as logic 0.												
15–12	SMOD	Source address modulo. Defines the size of the source data circular buffer used by the DMA Controller. If enabled (SMOD is non-zero), the buffer base address will be located on a boundary of the buffer size. The value of this boundary is based upon the initial source address (SAR). <table><tr><th>SMOD</th><th>Circular Buffer Size</th></tr><tr><td>0000</td><td>Buffer Disabled</td></tr><tr><td>0001</td><td>16 Bytes</td></tr><tr><td>0010</td><td>32 Bytes</td></tr><tr><td>...</td><td>...</td></tr><tr><td>1111</td><td>256 Kbytes</td></tr></table>	SMOD	Circular Buffer Size	0000	Buffer Disabled	0001	16 Bytes	0010	32 Bytes	...	...	1111	256 Kbytes
SMOD	Circular Buffer Size													
0000	Buffer Disabled													
0001	16 Bytes													
0010	32 Bytes													
...	...													
1111	256 Kbytes													
11–8	DMOD	Destination address modulo. Defines the size of the destination data circular buffer used by the DMA Controller. If enabled (DMOD value is non-zero), the buffer base address will be located on a boundary of the buffer size. The value of this boundary depends on the initial destination address (DAR). <table><tr><th>DMOD</th><th>Circular Buffer Size</th></tr><tr><td>0000</td><td>Buffer Disabled</td></tr><tr><td>0001</td><td>16 Bytes</td></tr><tr><td>0010</td><td>32 Bytes</td></tr><tr><td>...</td><td>...</td></tr><tr><td>1111</td><td>256 Kbytes</td></tr></table>	DMOD	Circular Buffer Size	0000	Buffer Disabled	0001	16 Bytes	0010	32 Bytes	...	...	1111	256 Kbytes
DMOD	Circular Buffer Size													
0000	Buffer Disabled													
0001	16 Bytes													
0010	32 Bytes													
...	...													
1111	256 Kbytes													
7	D_REQ	Disable request. DMA hardware automatically clears the corresponding DCR <sub>n</sub> [EEXT] bit when the byte count register reaches zero.  0 EEXT bit is not affected. 1 EEXT bit is cleared when the BCR is exhausted.												
6	—	Reserved, should be cleared.												



**Table 18-4. DCR<sub>n</sub> Field Descriptions (Continued)**

Bits	Name	Description
5–4	LINKCC	<p>Link channel control. Allows DMA channels to have their transfers linked. The current DMA channel will trigger a DMA request to the linked channels (LCH1 or LCH2) depending on the condition described by the LINKCC bits.</p> <p>00 No channel-to-channel linking  01 Perform a link to channel LCH1 after each cycle-steal transfer followed by a link to LCH2 after the BCR decrements to zero.  10 Perform a link to channel LCH1 after each cycle-steal transfer  11 Perform a link to channel LCH1 after the BCR decrements to zero</p> <p>If not in cycle steal mode (DCR<sub>n</sub>[CS]=0) and LINKCC=01 or 10, then no link to LCH1 will occur.</p> <p>If LINKCC = 01, a link to LCH1 is created after each cycle-steal transfer performed by the current DMA channel is completed. As the last cycle-steal is performed and the BCR reaches zero, then the link to LCH1 is closed and a link to LCH2 is created.</p> <p>If the LINKCC field is non-zero, the contents of the bandwidth control field (DCR<sub>n</sub>[BWC]) are ignored and effectively forced to zero by the DMA hardware. This is done to prevent any non-zero bandwidth control settings from allowing channel arbitration while any type of link is to be performed.</p>
3-2	LCH1	<p>Link channel 1. Indicates the DMA channel assigned as link channel 1. The link channel number cannot be the same as the currently executing channel, and generates a configuration error if this is attempted (DSR<sub>n</sub>[CE] is set).</p> <p>00 DMA Channel 0  01 DMA Channel 1  10 DMA Channel 2  11 DMA Channel 3</p>
1-0	LCH2	<p>Link channel 2. Indicates the DMA channel assigned as link channel 2. The link channel number cannot be the same as the currently executing channel, and generates a configuration error if this is attempted (DSR<sub>n</sub>[CE] is set).</p> <p>00 DMA Channel 0  01 DMA Channel 1  10 DMA Channel 2  11 DMA Channel 3</p>

## 18.4 Functional Description

In the following discussion, the term ‘DMA request’ implies that DCR<sub>n</sub>[START] or DCR<sub>n</sub>[EEXT] is set, followed by assertion of and internal or external DMA request. The START bit is cleared when the channel begins an internal access.

Before initiating a dual-address access, the DMA module verifies that DCR<sub>n</sub>[SSIZE,DSIZE] are consistent with the source and destination addresses. If they are not consistent, the configuration error bit, DSR<sub>n</sub>[CE], is set. If misalignment is detected, no transfer occurs, DSR<sub>n</sub>[CE] is set, and, depending on the DCR configuration, an interrupt event is issued. Note that if the auto-align bit, DCR<sub>n</sub>[AA], is set, error checking is performed on the appropriate registers.

A read/write transfer reads bytes from the source address and writes them to the destination address. The number of bytes is the larger of the sizes specified by  $DCRn[SSIZE]$  and  $DCRn[DSIZE]$ . See [Section 18.3.5, “DMA Control Registers \(DCR0–DCR3\).”](#)

Source and destination address registers ( $SARn$  and  $DARn$ ) can be programmed in the  $DCRn$  to increment at the completion of a successful transfer.  $BCRn$  decrements when an address transfer write completes for a single-address access ( $DCRn[SAA] = 0$ ) or when  $SAA = 1$ .

### 18.4.1 Transfer Requests (Cycle-Steal and Continuous Modes)

The DMA channel supports internal and external requests. A request is issued by setting  $DCRn[START]$  or by asserting  $\overline{DREQn}$ . Setting  $DCRn[EEXT]$  enables recognition of external DMA requests. Selecting between cycle-steal and continuous modes minimizes bus usage for either internal or external requests.

- Cycle-steal mode ( $DCRn[CS] = 1$ )—Only one complete transfer from source to destination occurs for each request. If  $DCRn[EEXT]$  is set, a request can be either internal or external. An internal request is selected by setting  $DCRn[START]$ . An external request is initiated by asserting  $\overline{DREQn}$  while  $DCRn[EEXT]$  is set. Note that multiple transfers will occur if  $\overline{DREQn}$  is continuously asserted.
- Continuous mode ( $DCRn[CS] = 0$ )—After an internal or external request, the DMA continuously transfers data until  $BCRn$  reaches zero or a multiple of  $DCRn[BWC]$  or until  $DSRn[DONE]$  is set. If  $BCRn$  is a multiple of  $BWC$ , the DMA request signal is negated until the bus cycle terminates to allow the internal arbiter to switch masters.  $DCRn[BWC] = 000$  specifies the maximum transfer rate; other values specify a transfer rate limit.

The DMA performs the specified number of transfers, then relinquishes bus control. The DMA negates its internal bus request on the last transfer before  $BCRn$  reaches a multiple of the boundary specified in  $BWC$ . On completion, the DMA reasserts its bus request to regain mastership at the earliest opportunity. The DMA loses bus control for a minimum of one bus cycle.

### 18.4.2 Dual-Address Data Transfer Mode

Each channel supports dual-address transfers. Dual-address transfers consist of a source data read and a destination data write. The DMA controller module begins a dual-address transfer sequence during a DMA request. If no error condition exists,  $DSRn[REQ]$  is set.

- Dual-address read—The DMA controller drives the  $SARn$  value onto the internal address bus. If  $DCRn[SINC]$  is set, the  $SARn$  increments by the appropriate number of bytes upon a successful read cycle. When the appropriate number of read cycles complete (multiple reads if the destination size is larger than the source), the DMA initiates the write portion of the transfer.

If a termination error occurs,  $DSRn[BES,DONE]$  are set and DMA transactions stop.

- Dual-address write—The DMA controller drives the  $DAR_n$  value onto the address bus. If  $DCR_n[DINC]$  is set,  $DAR_n$  increments by the appropriate number of bytes at the completion of a successful write cycle.  $BCR_n$  decrements by the appropriate number of bytes.  $DSR_n[DONE]$  is set when  $BCR_n$  reaches zero. If the  $BCR_n$  is greater than zero, another read/write transfer is initiated. If the  $BCR_n$  is a multiple of  $DCR_n[BWC]$ , the DMA request signal is negated until termination of the bus cycle to allow the internal arbiter to switch masters.

If a termination error occurs,  $DSR_n[BED,DONE]$  are set and DMA transactions stop.

### 18.4.3 Channel Initialization and Startup

Before a block transfer starts, channel registers must be initialized with information describing configuration, request-generation method, and the data block.

#### 18.4.3.1 Channel Prioritization

The four DMA channels are prioritized in ascending order (channel 0 having highest priority and channel 3 having the lowest) or in an order determined by  $DCR_n[BWC]$ . If the BWC encoding for a DMA channel is 000, that channel has priority only over the channel immediately preceding it. For example, if  $DCR3[BWC] = 000$ , DMA channel 3 has priority over DMA channel 2 (assuming  $DCR2[BWC] \neq 000$ ) but not over DMA channel 1.

If  $DCR0[BWC] = DCR1[BWC] = 000$ , DMA0 still has priority over DMA1. In this case,  $DCR1[BWC] = 000$  does not affect prioritization.

Simultaneous external requests are prioritized either in ascending order or in an order determined by each channel's  $DCR_n[BWC]$  bits.

#### 18.4.3.2 Programming the DMA Controller Module

Note the following general guidelines for programming the DMA:

- No mechanism exists within the DMA module itself to prevent writes to control registers during DMA accesses.
- If the  $DCR_n[BWC]$  value of sequential channels are equal, the channels are prioritized in ascending order.

The  $DMAREQC$  register is configured to assign peripheral DMA requests or external DMA request signals to the individual DMA channels.

The  $SAR_n$  is loaded with the source (read) address. If the transfer is from a peripheral device to memory, the source address is the location of the peripheral data register. If the transfer is from memory to either a peripheral device or memory, the source address is the starting address of the data block. This can be any aligned byte address.

The  $DAR_n$  should contain the destination (write) address. If the transfer is from a peripheral device to memory, or from memory to memory, the  $DAR_n$  is loaded with the starting address of the data block to be written. If the transfer is from memory to a peripheral device,  $DAR_n$  is loaded with the address of the peripheral data register. This address can be any aligned byte address.

$SAR_n$  and  $DAR_n$  change after each cycle depending on  $DCR_n[SSIZE,DSIZE,SINC,DINC,SMOD,DMOD]$  and on the starting address. Increment values can be 1, 2, 4, or 16 for byte, word, longword, or 16-byte line transfers, respectively. If the address register is programmed to remain unchanged (no count), the register is not incremented after the data transfer.

$BCR_n[BCR]$  must be loaded with the number of byte transfers to occur. It is decremented by 1, 2, 4, or 16 at the end of each transfer, depending on the transfer size.  $DSR_n[DONE]$  must be cleared for channel startup.

As soon as the channel has been initialized, it is started by writing a one to  $DCR_n[START]$  or asserting  $\overline{DREQ_n}$ , depending on the status of  $DCR_n[EEXT]$ . Programming the channel for internal requests causes the channel to request the bus and start transferring data immediately. If the channel is programmed for external request,  $\overline{DREQ_n}$  must be asserted before the channel requests the bus.

Changes to  $DCR_n$  are effective immediately while the channel is active. To avoid problems with changing a DMA channel setup, write a one to  $DSR_n[DONE]$  to stop the DMA channel.

## 18.4.4 Data Transfer

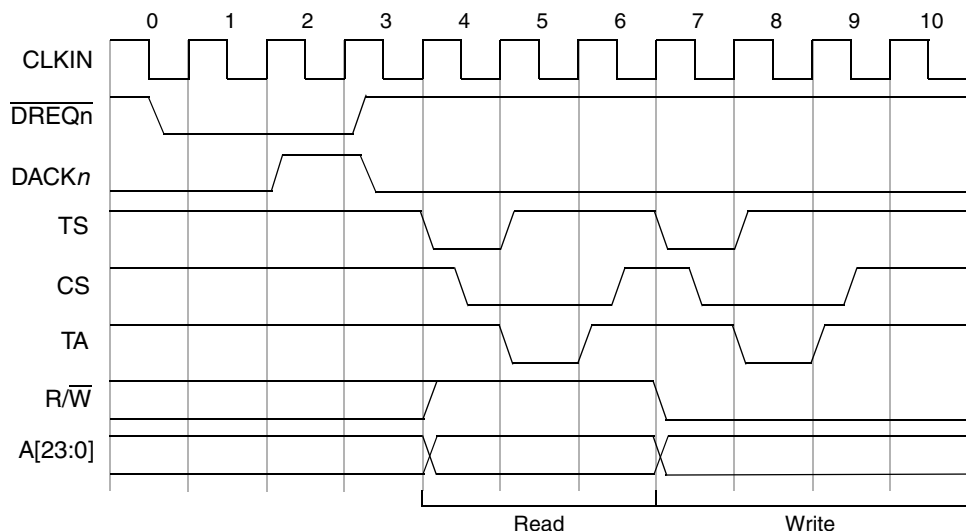
This section describes external requests, auto-alignment, and bandwidth control for DMA transfers.

### 18.4.4.1 External Request and Acknowledge Operation

The  $DMAREQC$  register in the System Control Module provides a software-controlled connection matrix between the on- and off-chip platform DMA request and acknowledge signals. Writing to this register determines the exact routing of the DMA requests to the four channels of the DMA module and the acknowledges back to the requesters. If  $DCR_n[EEXT]$  is set and the channel is idle, the assertion of the appropriate  $\overline{DREQ_n}$  or eTPU request activates channel  $n$ .

Channels 0, 1, and 2 initiate transfers to an external module by means of  $\overline{DREQ}[32:0]$ . (They are also available internally to the UART & DTIM interrupt signals.) The request for channel 3 is not connected externally and is only available internally to the eTPU, UART, and DTIM interrupt signals. If  $DCR_n[EEXT] = 1$  and the channel is idle, the DMA initiates a transfer when  $\overline{DREQ_n}$  is asserted or if the eTPU initiates a request.

Figure 18-9 shows the minimum 4-clock cycle delay from when  $\overline{\text{DREQ}}_n$  is sampled asserted to when a DMA bus cycle begins. This delay may be longer, depending on DMA priority, bus arbitration, and other factors. Figure 18-9 shows the relationship between the assertion of a properly enabled  $\overline{\text{DREQ}}_n$ , the DMA acknowledge, and the activation of the channel for a transfer where both the source and destination are mapped to chip select memories on the external bus.

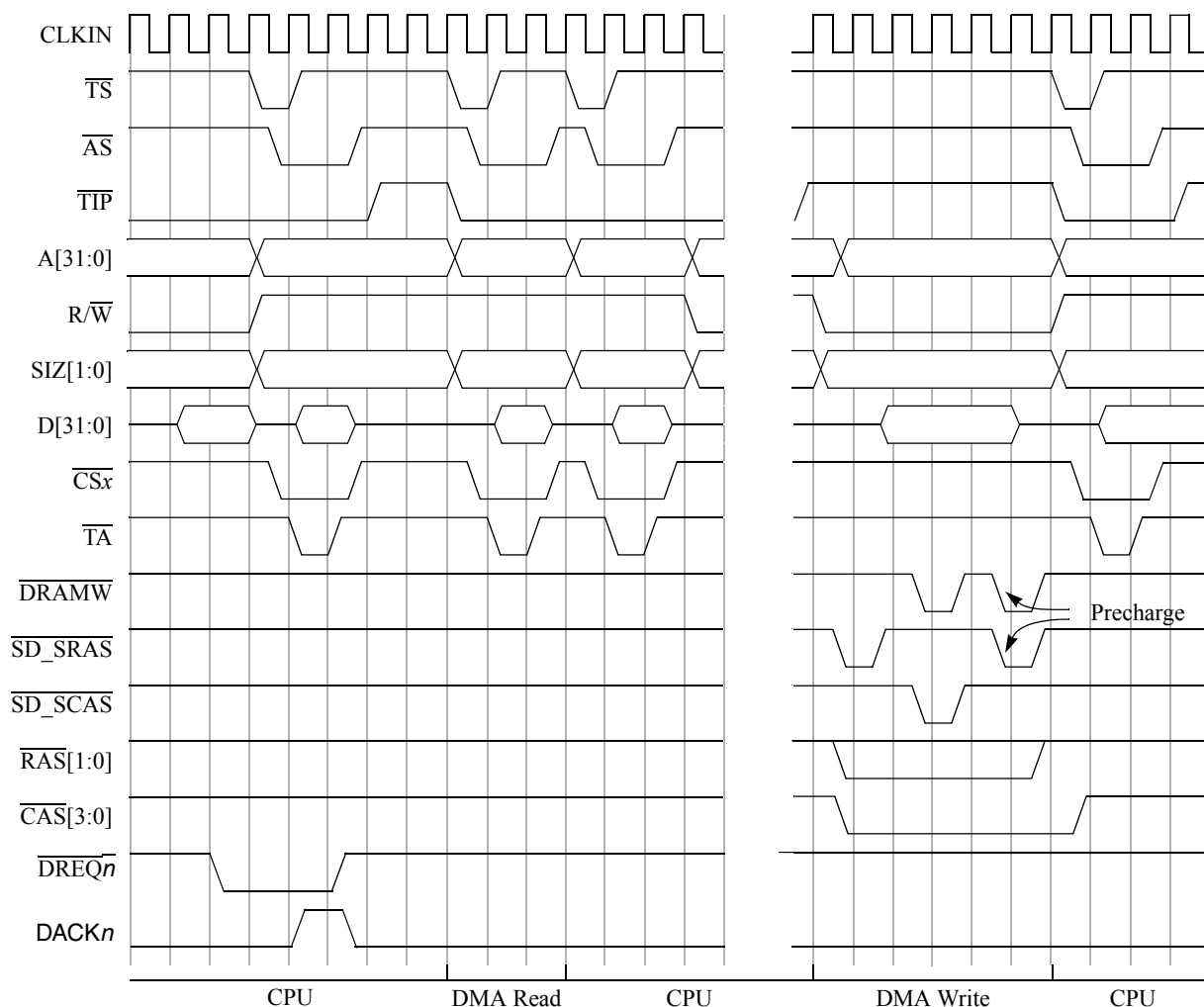


**Figure 18-9.  $\overline{\text{DREQ}}_n$  Timing Constraints, Dual-Address DMA Transfer**

Once the DMA module has detected the assertion of a properly-enabled  $\overline{\text{DREQ}}_n$ , it responds with a 1-cycle assertion of an acknowledge signal. This request/acknowledge handshake is provided so the request can be negated.

Since bus timings can vary from device to device, the diagrams below are conditionalized for 5210 only. The information is really just a repeat from the EIM/FlexBus and SDRAMC chapters, so the customers should not need this information. -MH

Figure 18-10 shows a dual-address, external peripheral-to-SDRAM DMA transfer. The DMA is not parked on the bus, so the diagram shows how the CPU can generate multiple bus cycles during DMA transfers. In cycle-steal mode, the maximum length of  $\overline{\text{DREQ}}$  assertion to maintain a single transfer is configuration-dependent. To avoid multiple transfers, for single-address accesses, no hold signal, byte accesses, and idle channels,  $\overline{\text{DREQ}}$  may be asserted for no more than four clock cycles.



**Figure 18-10. Dual-Address, Peripheral-to-SDRAM, Lower-Priority DMA Transfer**

Figure 18-11 shows a single-address DMA transfer in which an external peripheral is reading from memory.

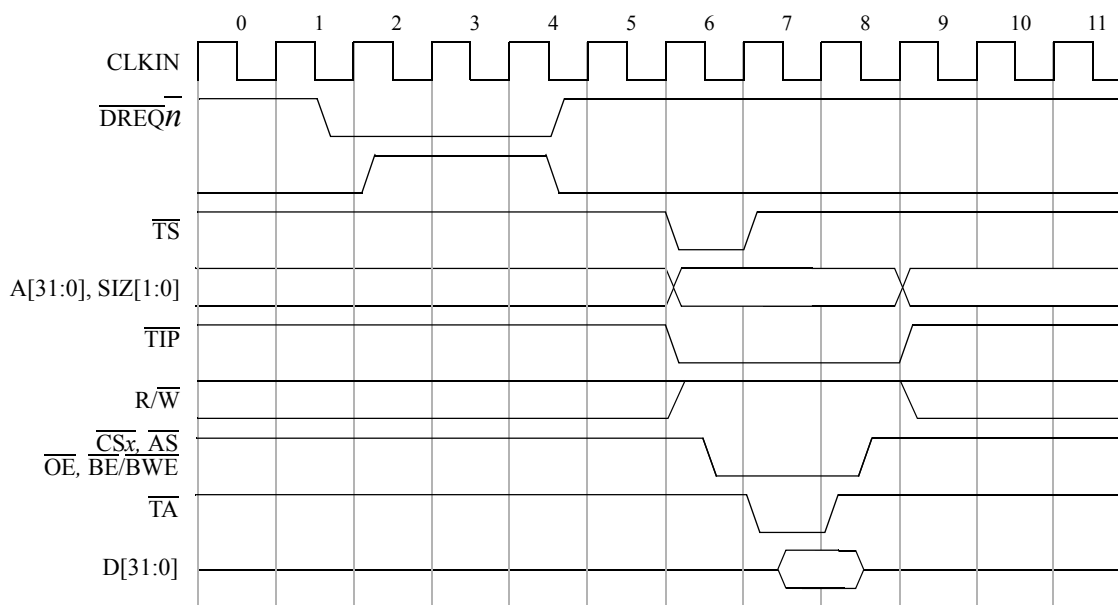


Figure 18-11. Single-Address DMA Transfer

#### 18.4.4.2 Auto-Alignment

Auto-alignment allows block transfers to occur at the optimal size based on the address, byte count, and programmed size. To use this feature,  $DCRn[AA]$  must be set. The source is auto-aligned if  $DCRn[SSIZE]$  indicates a transfer size larger than  $DCRn[DSIZE]$ . Source alignment takes precedence over the destination when the source and destination sizes are equal. Otherwise, the destination is auto-aligned. The address register chosen for alignment increments regardless of the increment value. Configuration error checking is performed on registers not chosen for alignment.

If  $BCRn$  is greater than 16, the address determines transfer size. Bytes, words, or longwords are transferred until the address is aligned to the programmed size boundary, at which time accesses begin using the programmed size.

If  $BCRn$  is less than 16 at the start of a transfer, the number of bytes remaining dictates transfer size. For example,  $AA = 1$ ,  $SARn = 0x0001$ ,  $BCRn = 0x00F0$ ,  $SSIZE = 00$  (longword), and  $DSIZE = 01$  (byte). Because  $SSIZE > DSIZE$ , the source is auto-aligned. Error checking is performed on destination registers. The access sequence is as follows:

1. Read byte from  $0x0001$ —write 1 byte, increment  $SARn$ .
2. Read word from  $0x0002$ —write 2 bytes, increment  $SARn$ .
3. Read longword from  $0x0004$ —write 4 bytes, increment  $SARn$ .
4. Repeat longwords until  $SARn = 0x00F0$ .
5. Read byte from  $0x00F0$ —write byte, increment  $SARn$ .

If DSIZE is another size, data writes are optimized to write the largest size allowed based on the address, but not exceeding the configured size.

### 18.4.4.3 Bandwidth Control

Bandwidth control makes it possible to force the DMA off the bus to allow access to another device.  $DCR_n[BWC]$  provides seven levels of block transfer sizes. If the  $BCR_n$  decrements to a multiple of the decode of the BWC, the DMA bus request negates until the bus cycle terminates. If a request is pending, the arbiter may then pass bus mastership to another device. If auto-alignment is enabled,  $DCR_n[AA] = 1$ , the  $BCR_n$  may skip over the programmed boundary, in which case, the DMA bus request is not negated.

If  $BWC = 000$ , the request signal remains asserted until  $BCR_n$  reaches zero. DMA has priority over the core. Note that in this scheme, the arbiter can always force the DMA to relinquish the bus. See [Section 11.3.3, “Bus Master Park Register \(MPARK\).”](#)

### 18.4.5 Termination

An unsuccessful transfer can terminate for one of the following reasons:

- Error conditions—When the MCF5275 encounters a read or write cycle that terminates with an error condition,  $DSR_n[BES]$  is set for a read and  $DSR_n[BED]$  is set for a write before the transfer is halted. If the error occurred in a write cycle, data in the internal holding register is lost.
- Interrupts—If  $DCR_n[INT]$  is set, the DMA drives the appropriate internal interrupt signal. The processor can read  $DSR_n$  to determine whether the transfer terminated successfully or with an error.  $DSR_n[DONE]$  is then written with a one to clear the interrupt and the DONE and error bits.



# Chapter 19

## Fast Ethernet Controllers (FEC0 & FEC1)

### 19.1 Introduction

This chapter 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.

#### 19.1.1 Overview

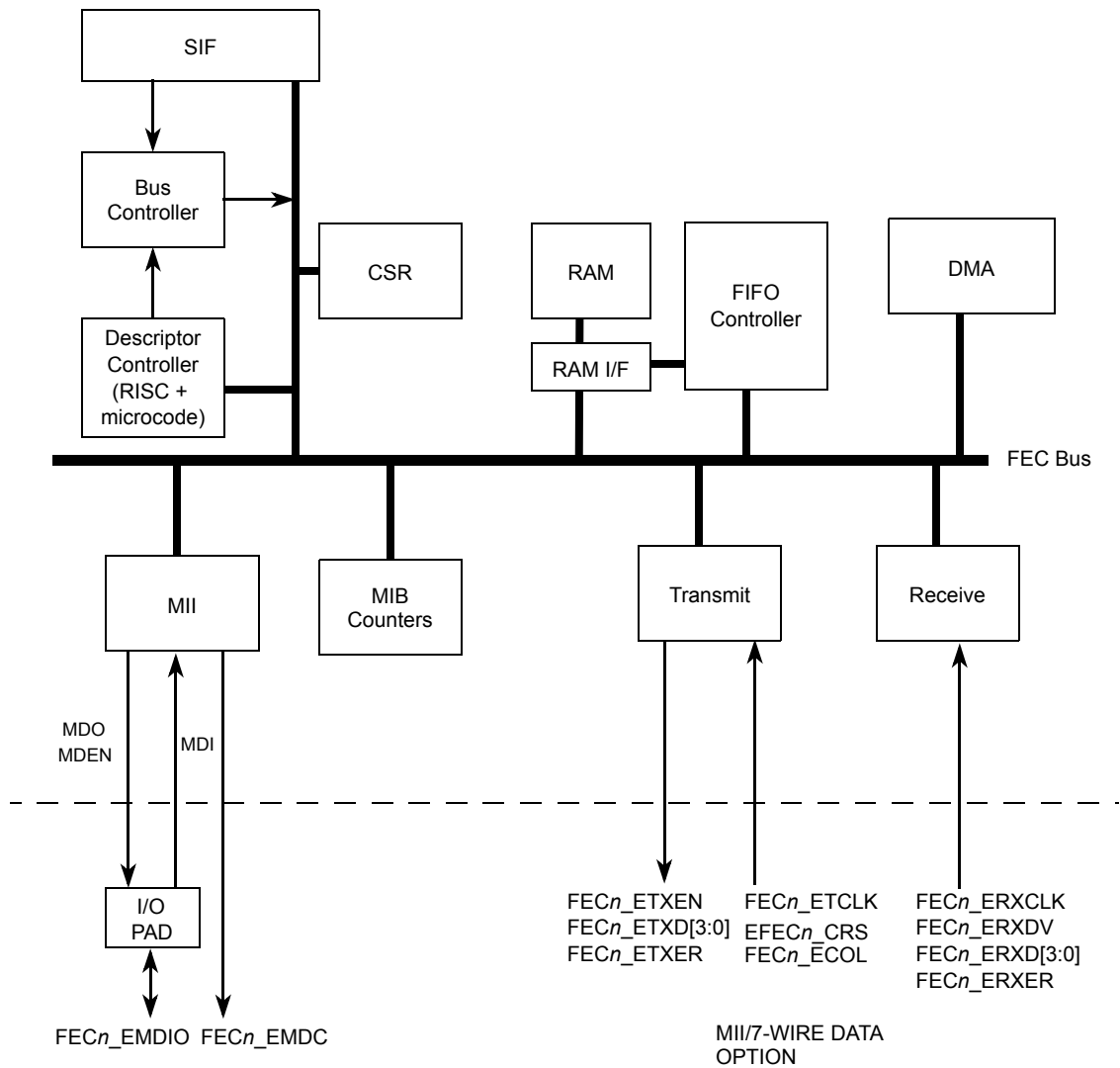
The Ethernet Media Access Controller (MAC) is designed to support both 10 and 100 Mbps Ethernet/IEEE 802.3 networks. An external transceiver interface and transceiver function are required to complete the interface to the media. The FECs support three different standard MAC-PHY (physical) interfaces for connection to an external Ethernet transceiver. The FECs support the 10/100 Mbps MII and the 10 Mbps-only 7-wire interface, which uses a subset of the MII pins.

#### NOTE

The GPIO module must be configured to enable the peripheral function of the appropriate pins (refer to [Chapter 12, “General Purpose I/O Module”](#)) prior to configuring the FECs.

#### 19.1.2 Block Diagram

The block diagram of a single FEC is shown below. The FECs are implemented with a combination of hardware and microcode. The off-chip (Ethernet) interfaces are compliant with industry and IEEE 802.3 standards.



**Figure 19-1. FECn Block Diagram**

The descriptor controller is a RISC-based controller that provides the following functions in the FECs:

- 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 18, “DMA Controller Module,”](#) nor to the DMA timers described in [Chapter 25, “DMA Timers \(DTIM0–DTIM3\).”](#)

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  $FRSR_n$  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 FECs by writing, through the SIF (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  $FEC_n\_EMDC$  (Management Data Clock) and  $FEC_n\_EMDIO$  (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. See [Section 19.2.3, “MIB Block Counters Memory Map”](#) for more information.

### 19.1.3 Features

The FECs incorporate 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)
- IEEE 802.3 full duplex flow control
- Programmable max frame length supports IEEE 802.1 VLAN tags and priority

- Support for full-duplex operation (200Mbps throughput) with a minimum system clock rate of 50 MHz
- Support for half-duplex operation (100Mbps throughput) with a minimum system clock rate of 25 MHz
- Retransmission from transmit FIFO following a collision (no processor bus utilization)
- Automatic internal flushing of the receive FIFO for runts (collision fragments) and address recognition rejects (no processor 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

## 19.1.4 Modes of Operation

The primary operational modes are described in this section.

### 19.1.4.1 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  $TCRn[FDEN]$ .

When configured for full duplex mode, flow control may be enabled. Refer to the  $TCRn[RFC\_PAUSE]$  and  $TCRn[TFC\_PAUSE]$  bits, the  $RCRn[FCE]$  bit, and [Section 19.3.10, “Full Duplex Flow Control,”](#) for more details.

## 19.1.5 Interface Options

The following interface options are supported. A detailed discussion of the interface configurations is provided in [Section 19.3.5, “Network Interface Options”](#).

### 19.1.5.1 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  $RCRn[MII\_MODE]$ .

The speed of operation is determined by the  $FECn\_ETXCLK$  and  $FECn\_ERXCLK$  pins which are driven by the external transceiver. The transceiver will either auto-negotiate the speed or it may be

controlled by software via the serial management interface (FEC $n$ \_EMDC/FEC $n$ \_EMDIO pins) to the transceiver. Refer to the MMFR $n$  and MSCR $n$  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.

### 19.1.5.2 10 Mbps 7-Wire Interface Operation

The FECs support a 7-wire interface as used by many 10 Mbps ethernet transceivers. The RCR[MII\_MODE] bit controls this functionality. If this bit is cleared, the MII mode is disabled and the 10 Mbps, 7-wire mode is enabled.

### 19.1.6 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 19.3.8, “Ethernet Address Recognition.”](#)

### 19.1.7 Internal Loopback

Internal loopback mode is selected via RCR $n$ [LOOP]. Loopback mode is discussed in detail in [Section 19.3.13, “Internal and External Loopback.”](#)

## 19.2 Memory Map/Register Definition

This section gives an overview of the registers, followed by a description of the buffers.

The FECs are 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.

### 19.2.1 High-Level Module Memory Map

Each FEC implementation requires a 1-Kbyte memory map space. This is divided into 2 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 19-1](#) defines the top level memory map.

**Table 19-1. Module Memory Map**

Address	Function
IPSBAR + 0x1000–11FF	FEC0 Control/Status Registers
IPSBAR + 0x1200–13FF	FEC0 MIB Block Counters
IPSBAR + 0x1800–19FF	FEC1 Control/Status Registers
IPSBAR + 0x2000–21FF	FEC1 MIB Block Counters

## 19.2.2 Register Memory Map

Table 19-2 shows the FEC register memory map with each register address, name, and a brief description.

**Table 19-2. FEC Register Memory Map**

IPSBAR Offset (FEC0 & FEC1)	Name	Description	Size (bits)
0x1004 & 0x1804	EIR <sub>n</sub>	Interrupt Event Register	32
0x1008 & 0x1808	EIMR <sub>n</sub>	Interrupt Mask Register	32
0x1010 & 0x1810	RDAR <sub>n</sub>	Receive Descriptor Active Register	32
0x1014 & 0x1814	TDAR <sub>n</sub>	Transmit Descriptor Active Register	32
0x1024 & 0x1824	ECR <sub>n</sub>	Ethernet Control Register	32
0x1040 & 0x1840	MDAT <sub>n</sub>	MII Data Register	32
0x1044 & 0x1844	MSCR <sub>n</sub>	MII Speed Control Register	32
0x1064 & 0x1864	MIBC <sub>n</sub>	MIB Control/Status Register	32
0x1084 & 0x1884	RCR <sub>n</sub>	Receive Control Register	32
0x10C4 & 0x18C4	TCR <sub>n</sub>	Transmit Control Register	32
0x10E4 & 0x18E4	PALR <sub>n</sub>	Physical Address Low Register	32
0x10E8 & 0x18E8	PAUR <sub>n</sub>	Physical Address High+ Type Field	32
0x10EC & 0x18EC	OPD <sub>n</sub>	Opcode + Pause Duration	32
0x1118 & 0x1918	IAUR <sub>n</sub>	Upper 32 bits of Individual Hash Table	32
0x111C & 0x191C	IALR <sub>n</sub>	Lower 32 Bits of Individual Hash Table	32
0x1120 & 0x1920	GAUR <sub>n</sub>	Upper 32 bits of Group Hash Table	32
0x1124 & 0x1924	GALR <sub>n</sub>	Lower 32 bits of Group Hash Table	32
0x1144 & 0x1944	TFWR <sub>n</sub>	Transmit FIFO Watermark	32
0x114C & 0x194C	FRBR <sub>n</sub>	FIFO Receive Bound Register	32
0x1150 & 0x1950	FRSR <sub>n</sub>	FIFO Receive FIFO Start Registers	32
0x1180 & 0x1980	ERDSR <sub>n</sub>	Pointer to Receive Descriptor Ring	32
0x1184 & 0x1984	ETDSR <sub>n</sub>	Pointer to Transmit Descriptor Ring	32
0x1188 & 0x1988	EMRBR <sub>n</sub>	Maximum Receive Buffer Size	32

## 19.2.3 MIB Block Counters Memory Map

Table 19-3 defines the MIB Counters memory map which defines the locations in the MIB RAM space where hardware maintained counters reside. These fall in the 0x1200-0x13FF & 0x2000-0x21FF 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 FECs only support frame lengths up to 2032 bytes. The RMON counters are implemented independently for transmit and receive to insure 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 FECs 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 19-3. MIB Counters Memory Map**

IPSBAR Offset	Mnemonic	Description
0x1200 & 0x2000	RMON_T_DROP $n$	Count of frames not counted correctly
0x1204 & 0x2004	RMON_T_PACKETS $n$	RMON Tx packet count
0x1208 & 0x2008	RMON_T_BC_PKT $n$	RMON Tx Broadcast Packets
0x120C & 0x200C	RMON_T_MC_PKT $n$	RMON Tx Multicast Packets
0x1210 & 0x2010	RMON_T_CRC_ALIGN $n$	RMON Tx Packets w CRC/Align error
0x1214 & 0x2014	RMON_T_UNDERSIZE $n$	RMON Tx Packets < 64 bytes, good crc
0x1218 & 0x2018	RMON_T_OVERSIZE $n$	RMON Tx Packets > MAX_FL bytes, good crc
0x121C & 0x201C	RMON_T_FRAG $n$	RMON Tx Packets < 64 bytes, bad crc
0x1220 & 0x2020	RMON_T_JAB $n$	RMON Tx Packets > MAX_FL bytes, bad crc
0x1224 & 0x2024	RMON_T_COL $n$	RMON Tx collision count
0x1228 & 0x2028	RMON_T_P64 $n$	RMON Tx 64 byte packets
0x122C & 0x202C	RMON_T_P65TO127 $n$	RMON Tx 65 to 127 byte packets
0x1230 & 0x2030	RMON_T_P128TO255 $n$	RMON Tx 128 to 255 byte packets
0x1234 & 0x2034	RMON_T_P256TO511 $n$	RMON Tx 256 to 511 byte packets
0x1238 & 0x2038	RMON_T_P512TO1023 $n$	RMON Tx 512 to 1023 byte packets
0x123C & 0x203C	RMON_T_P1024TO2047 $n$	RMON Tx 1024 to 2047 byte packets
0x1240 & 0x2040	RMON_T_P_GTE2048 $n$	RMON Tx packets w > 2048 bytes
0x1244 & 0x2044	RMON_T_OCTETS $n$	RMON Tx Octets
0x1248 & 0x2048	IEEE_T_DROP $n$	Count of frames not counted correctly
0x124C & 0x204C	IEEE_T_FRAME_OK $n$	Frames Transmitted OK
0x1250 & 0x2050	IEEE_T_1COL $n$	Frames Transmitted with Single Collision
0x1254 & 0x2054	IEEE_T_MCOL $n$	Frames Transmitted with Multiple Collisions
0x1258 & 0x2058	IEEE_T_DEF $n$	Frames Transmitted after Deferral Delay
0x125C & 0x205C	IEEE_T_LCOL $n$	Frames Transmitted with Late Collision
0x1260 & 0x2060	IEEE_T_EXCOL $n$	Frames Transmitted with Excessive Collisions

**Table 19-3. MIB Counters Memory Map (Continued)**

IPSBAR Offset	Mnemonic	Description
0x1264 & 0x2064	IEEE_T_MACERR $n$	Frames Transmitted with Tx FIFO Underrun
0x1268 & 0x2068	IEEE_T_CSERR $n$	Frames Transmitted with Carrier Sense Error
0x126C & 0x206C	IEEE_T_SQEN $n$	Frames Transmitted with SQE Error
0x1270 & 0x2070	IEEE_T_FDXFC $n$	Flow Control Pause frames transmitted
0x1274 & 0x2074	IEEE_T_OCTETS_OK $n$	Octet count for Frames Transmitted w/o Error
0x1284 & 0x2084	RMON_R_PACKETS $n$	RMON Rx packet count
0x1288 & 0x2088	RMON_R_BC_PKT $n$	RMON Rx Broadcast Packets
0x128C & 0x208C	RMON_R_MC_PKT $n$	RMON Rx Multicast Packets
0x1290 & 0x2090	RMON_R_CRC_ALIGN $n$	RMON Rx Packets w CRC/Align error
0x1294 & 0x2094	RMON_R_UNDERSIZE $n$	RMON Rx Packets < 64 bytes, good crc
0x1298 & 0x2098	RMON_R_OVERSIZE $n$	RMON Rx Packets > MAX_FL bytes, good crc
0x129C & 0x209C	RMON_R_FRAG $n$	RMON Rx Packets < 64 bytes, bad crc
0x12A0 & 0x21A0	RMON_R_JAB $n$	RMON Rx Packets > MAX_FL bytes, bad crc
0x12A4 & 0x21A4	RMON_R_RESVD_0 $n$	
0x12A8 & 0x21A8	RMON_R_P64 $n$	RMON Rx 64 byte packets
0x12AC & 0x21AC	RMON_R_P65TO127 $n$	RMON Rx 65 to 127 byte packets
0x12B0 & 0x21B0	RMON_R_P128TO255 $n$	RMON Rx 128 to 255 byte packets
0x12B4 & 0x21B4	RMON_R_P256TO511 $n$	RMON Rx 256 to 511 byte packets
0x12B8 & 0x21B8	RMON_R_P512TO1023 $n$	RMON Rx 512 to 1023 byte packets
0x12BC & 0x21BC	RMON_R_P1024TO2047 $n$	RMON Rx 1024 to 2047 byte packets
0x12C0 & 0x21C0	RMON_R_P_GTE2048 $n$	RMON Rx packets w > 2048 bytes
0x12C4 & 0x21C4	RMON_R_OCTETS $n$	RMON Rx Octets
0x12C8 & 0x21C8	IEEE_R_DROP $n$	Count of frames not counted correctly
0x12CC & 0x21CC	IEEE_R_FRAME_OK $n$	Frames Received OK
0x12D0 & 0x21D0	IEEE_R_CRC $n$	Frames Received with CRC Error
0x12D4 & 0x21D4	IEEE_R_ALIGN $n$	Frames Received with Alignment Error
0x12D8 & 0x21D8	IEEE_R_MACERR $n$	Receive Fifo Overflow count
0x12DC & 0x21DC	IEEE_R_FDXFC $n$	Flow Control Pause frames received
0x12E0 & 0x21E0	IEEE_R_OCTETS_OK $n$	Octet count for Frames Rcvd w/o Error

## 19.2.4 Register Description

The following sections describe each register in detail.



### 19.2.4.1 Ethernet Interrupt Event Register (EIR)

When an event occurs that sets a bit in the  $EIR_n$ , an interrupt will be 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 upon 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 will be 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 - IEEE\_T\_MACERR

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	HBERR	BABR	BABT	GRA	TXF	TXB	RXF	RXB	MII	EBERR	LC	RL	UN	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
Address	IPSBAR + 0x1004 (FEC0) & IPSBAR + 0x1804 (FEC1)															

**Figure 19-2. Ethernet Interrupt Event Register (EIR<sub>n</sub>)**

**Table 19-4. EIR<sub>n</sub> Field Descriptions**

Bits	Name	Description
31	HBERR	Heartbeat error. This interrupt indicates that HBC is set in the TCR <sub>n</sub> register and that the COL input was not asserted within the Heartbeat window following a transmission.
30	BABR	Babbling receive error. This bit indicates a frame was received with length in excess of RCR <sub>n</sub> [MAX_FL] bytes.

**Table 19-4. EIR<sub>n</sub> Field Descriptions (Continued)**

Bits	Name	Description
29	BABT	Babbling transmit error. This bit indicates that the transmitted frame length has exceeded RCR <sub>n</sub> [MAX_FL] bytes. This condition is usually caused by a frame that is too long being placed into the transmit data buffer(s). Truncation does not occur.
28	GRA	Graceful stop complete. This interrupt will be 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. 1) A graceful stop, which was initiated by the setting of the TCR <sub>n</sub> [GTS] bit is now complete. 2) A graceful stop, which was initiated by the setting of the TCR <sub>n</sub> [TFC_PAUSE] bit is now complete. 3) A graceful stop, which was initiated by the reception of a valid full duplex flow control "pause" frame is now complete. Refer to <a href="#">Section 19.3.10, "Full Duplex Flow Control."</a>
27	TXF	Transmit frame interrupt. This bit indicates that a frame has been transmitted and that the last corresponding buffer descriptor has been updated.
26	TXB	Transmit buffer interrupt. This bit indicates that a transmit buffer descriptor has been updated.
25	RXF	Receive frame interrupt. This bit indicates that a frame has been received and that the last corresponding buffer descriptor has been updated.
24	RXB	Receive buffer interrupt. This bit indicates that a receive buffer descriptor has been updated that was not the last in the frame.
23	MII	MII interrupt. This bit indicates that the MII has completed the data transfer requested.
22	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 <sub>n</sub> [ETHER_EN] will be cleared, halting frame processing by the FEC. When this occurs software will need to insure that the FIFO controller and DMA are also soft reset.
21	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.
20	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 will commence. Can only occur in half duplex mode.
19	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.
18–0	—	Reserved, should be cleared.

#### 19.2.4.2 Interrupt Mask Registers (EIMR0 & EIMR1)

The EIMR<sub>n</sub> registers control which interrupt events are allowed to generate actual interrupts. All implemented bits in this CSR are read/write. This register is cleared upon a hardware reset. If the corresponding bits in both the EIR<sub>n</sub> and EIMR<sub>n</sub> registers are set, the interrupt will be signalled to the CPU. The interrupt signal will remain asserted until a 1 is written to the EIR<sub>n</sub> bit (write 1 to clear) or a 0 is written to the EIMR<sub>n</sub> bit.

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	HB ERR	BABR	BABT	GRA	TXF	TXB	RXF	RXB	MII	EB ERR	LC	RL	UN	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
Address	IPSBAR + 0x1008 (FEC0) & IPSBAR + 0x1808 (FEC1)															

**Figure 19-3. Interrupt Mask Register (EIMR<sub>n</sub>)****Table 19-5. EIMR<sub>n</sub> Field Descriptions**

Bits	Name	Description
31–19	See <a href="#">Figure 19-3</a> and <a href="#">Table 19-4</a> .	Interrupt mask. Each bit corresponds to an interrupt source defined by the EIR <sub>n</sub> register. The corresponding EIMR <sub>n</sub> bit determines whether an interrupt condition can generate an interrupt. At every processor clock, the EIR <sub>n</sub> samples the signal generated by the interrupting source. The corresponding EIR <sub>n</sub> bit reflects the state of the interrupt signal even if the corresponding EIMR <sub>n</sub> bit is set. 0 The corresponding interrupt source is masked. 1 The corresponding interrupt source is not masked.
18–0	—	Reserved, should be cleared.

### 19.2.4.3 Receive Descriptor Active Registers (RDAR0 & RDAR1)

RDAR<sub>n</sub> 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 RDAR bit is set. This is independent of the data actually written by the user. When set, the FEC will poll the receive descriptor ring and process receive frames (provided ECR<sub>n</sub>[ETHER\_EN] is also set). Once the FEC polls a receive descriptor whose empty bit is not set, then the FEC will clear the RDAR bit and cease 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 registers are cleared at reset and when ECR<sub>n</sub>[ETHER\_EN] is cleared.

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	RDAR	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
Address	IPSBAR + 0x1010 (FEC0) & IPSBAR + 0x1810 (FEC1)															

**Figure 19-4. Receive Descriptor Active Register (RDAR<sub>n</sub>)****Table 19-6. RDAR<sub>n</sub> Field Descriptions**

Bits	Name	Description
31–25	—	Reserved, should be cleared.
24	RDAR	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 <sub>n</sub> [ETHER_EN] is cleared.
23–0	—	Reserved, should be cleared.

#### 19.2.4.4 Transmit Descriptor Active Registers (TDAR0 & TDAR1)

The TDAR<sub>n</sub> are command registers which 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 TDAR bit is set. This value is independent of the data actually written by the user. When set, the FEC will poll the transmit descriptor ring and process transmit frames (provided ECR<sub>n</sub>[ETHER\_EN] is also set). Once the FEC polls a transmit descriptor whose ready bit is not set, then the FEC will clear the TDAR bit and cease transmit descriptor ring polling until the user sets the bit again, signifying additional descriptors have been placed into the transmit descriptor ring.

The TDAR<sub>n</sub> register is cleared at reset, when ECR<sub>n</sub>[ETHER\_EN] is cleared, or when ECR<sub>n</sub>[RESET] is set.

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	TDAR	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
Address	IPSBAR + 0x1014 (FEC0) & IPSBAR + 0x1814 (FEC1)															

Figure 19-5. Transmit Descriptor Active Register (TDAR<sub>n</sub>)Table 19-7. TDAR<sub>n</sub> Field Descriptions

Bits	Name	Description
31–25	—	Reserved, should be cleared.
24	TDAR	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 <sub>n</sub> [ETHER_EN] is cleared.
23–0	—	Reserved, should be cleared.

### 19.2.4.5 Ethernet Control Registers (ECR0 & ECR1)

ECR<sub>n</sub> is a read/write user register, though both fields in this register may be altered by hardware as well. The ECR<sub>n</sub> is used to enable/disable the FEC.

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
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
	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	ETHER_EN	RESET
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Address	IPSBAR + 0x1024															

Figure 19-6. Ethernet Control Register (ECR<sub>n</sub>)Table 19-8. ECR<sub>n</sub> Field Descriptions

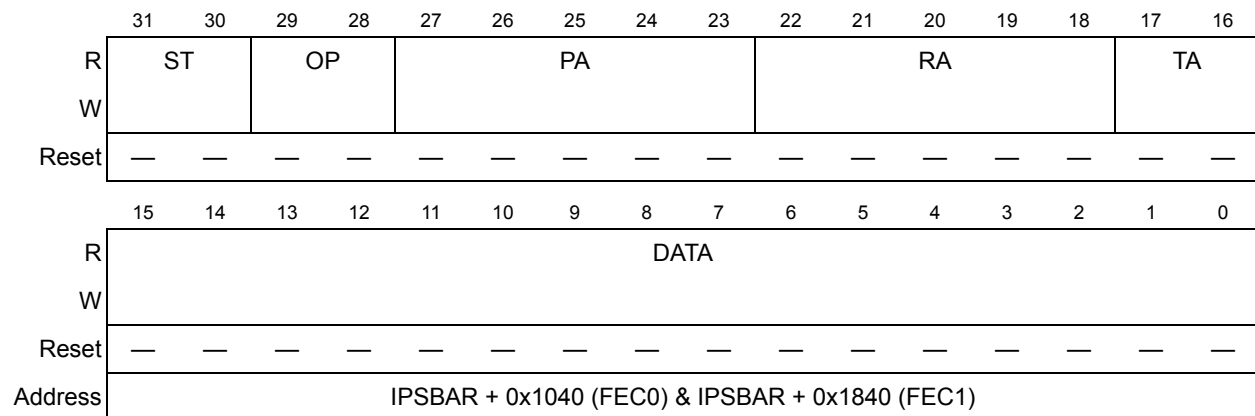
Bits	Name	Description
31–2	—	Reserved.

**Table 19-8. ECR<sub>n</sub> Field Descriptions (Continued)**

Bits	Name	Description
1	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 descriptor(s) 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: <ul style="list-style-type: none"> <li>• ECR<sub>n</sub>[RESET] is set by software, in which case ETHER_EN will be cleared</li> <li>• An error condition causes the EIR<sub>n</sub>[EBERR] bit to set, in which case ETHER_EN will be cleared</li> </ul>
0	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.

#### 19.2.4.6 MII Management Frame Registers (MMFR0 & MMFR1)

The MMFR<sub>n</sub> is accessed by the user and does not reset to a defined value. The MMFR<sub>n</sub> registers are used to communicate with the attached MII compatible PHY device(s), providing read/write access to their MII registers. Performing a write to the MMFR<sub>n</sub> will cause a management frame to be sourced unless the MSCR<sub>n</sub> has been programmed to 0. In the case of writing to MMFR<sub>n</sub> when MSCR<sub>n</sub> = 0, if the MSCR<sub>n</sub> register is then written to a non-zero value, an MII frame will be generated with the data previously written to the MMFR<sub>n</sub>. This allows MMFR<sub>n</sub> and MSCR<sub>n</sub> to be programmed in either order if MSCR<sub>n</sub> is currently zero.

**Figure 19-7. MII Management Frame Register (MMFR<sub>n</sub>)**

**Table 19-9. MMFR<sub>n</sub> Field Descriptions**

Bit	Name	Description
31–30	ST	Start of frame delimiter. These bits must be programmed to 01 for a valid MII management frame.
29–28	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 will produce “read” frame operation while a value of 00 will produce “write” frame operation, but these frames will not be MII compliant.
27–23	PA	PHY address. This field specifies one of up to 32 attached PHY devices.
22–18	RA	Register address. This field specifies one of up to 32 registers within the specified PHY device.
17–16	TA	Turn around. This field must be programmed to 10 to generate a valid MII management frame.
15–0	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<sub>n</sub> 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 will be generated but will 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<sub>n</sub> register. Writing this pattern will cause the control logic to shift out the data in the MMFR<sub>n</sub> register following a preamble generated by the control state machine. During this time the contents of the MMFR<sub>n</sub> register will be altered as the contents are serially shifted and will be unpredictable if read by the user. Once the write management frame operation has completed, the MII interrupt will be generated. At this time the contents of the MMFR<sub>n</sub> register will 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<sub>n</sub> register (the content of the DATA field is a don't care). Writing this pattern will cause the control logic to shift out the data in the MMFR<sub>n</sub> register following a preamble generated by the control state machine. During this time the contents of the MMFR<sub>n</sub> register will be altered as the contents are serially shifted, and will be unpredictable if read by the user. Once the read management frame operation has completed, the MII interrupt will be generated. At this time the contents of the MMFR<sub>n</sub> register will 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<sub>n</sub> register is written while frame generation is in progress, the frame contents will be altered. Software should use the MII\_STATUS<sub>n</sub> register and/or the MII interrupt to avoid writing to the MMFR<sub>n</sub> register while frame generation is in progress.

### 19.2.4.7 MII Speed Control Registers (MSCR0 & MSCR1)

The  $MSCR_n$  provides control of the MII clock ( $FEC_n\_EMDC$  pin) frequency and allows a preamble drop on the MII management frame.

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
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
	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	DIS_PRE	MII_SPEED						0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Address	IPSBAR + 0x1044 (FEC0) & IPSBAR + 0x1844 (FEC1)															

**Figure 19-8. MII Speed Control Register (MSCRn)**

**Table 19-10. MSCR Field Descriptions**

Bits	Name	Description
31–8	—	Reserved, should be cleared.
7	DIS_PRE	Asserting this bit will cause 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 device(s) does not require it.
6–1	MII_SPEED	MII_SPEED controls the frequency of the MII management interface clock ( $FEC_n\_EMDC$ ) relative to the system clock. A value of 0 in this field will “turn off” the $FEC_n\_EMDC$ and leave it in low voltage state. Any non-zero value will result in the $FEC_n\_EMDC$ frequency of $1/(MII\_SPEED \times 2)$ of the system clock frequency.
0	—	Reserved, should be cleared.

The MII\_SPEED field must be programmed with a value to provide an  $FEC_n\_EMDC$  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  $MSCR_n$  register may optionally be set to zero to turn off the  $FEC_n\_EMDC$ . The  $FEC_n\_EMDC$  generated will have a 50% duty cycle except when MII\_SPEED is changed during operation (change will take effect following either a rising or falling edge of  $FEC_n\_EMDC$ ).

If the system clock is 83 MHz, programming MII\_SPEED to 0x11 will result in an  $FEC_n\_EMDC$  frequency of  $83 \text{ MHz} / (17 \times 2) = 2.44 \text{ MHz}$ . A table showing optimum values for MII\_SPEED as a function of system clock frequency is provided below.



**Table 19-11. Programming Examples for MSCR<sub>n</sub>**

System Clock Frequency	MSCR[MII_SPEED]	FEC <sub>n</sub> _EMDC frequency
25 MHz	0x5	2.5 MHz
33 MHz	0x7	2.36 MHz
40 MHz	0x8	2.5 MHz
50 MHz	0xA	2.5 MHz
66 MHz	0xE	2.36 MHz
75 MHz	0xF	2.5 MHz
83 MHz	0x11	2.44 MHz

### 19.2.4.8 MIB Control Registers (MIBC0 & MIBC1)

The MIBC<sub>n</sub> 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\_DIS bit is reset to 1. See [Table 19-3](#) for the locations of the MIB counters.

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	MIB_DIS	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
	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
Address	IPSBAR + 0x1064 (FEC0) & IPSBAR + 0x1864 (FEC1)															

**Figure 19-9. MIB Control Register (MIBC<sub>n</sub>)****Table 19-12. MIBC<sub>n</sub> Field Descriptions**

Bits	Name	Description
31	MIB_DIS	A read/write control bit. If set, the MIB logic will halt and not update any MIB counters.
30	MIB_IDLE	A read-only status bit. If set the MIB block is not currently updating any MIB counters.
29–0	—	Reserved.

### 19.2.4.9 Receive Control Registers (RCR0 & RCR1)

The  $RCR_n$ , programmed by the user, controls the operational mode of the receive block and should be written only when  $ECR_n[ETHER\_EN] = 0$  (initialization time).

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
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
	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	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	1
Address	IPSBAR + 0x1084 (FEC0) & IPSBAR + 0x1884 (FEC1)															

**Figure 19-10. Receive Control Register (RCR)**

**Table 19-13. RCR Field Descriptions**

Bits	Name	Description
31–27	—	Reserved, should be cleared.
26–16	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 will cause the BABT interrupt to occur. Receive Frames longer than MAX_FL will cause the BABR interrupt to occur and will set the LG bit in the end of frame receive buffer descriptor. The recommended default value to be programmed by the user is 1518 or 1522 (if VLAN Tags are supported).
15–6	—	Reserved, should be cleared.
5	FCE	Flow control enable. If asserted, the receiver will detect PAUSE frames. Upon PAUSE frame detection, the transmitter will stop transmitting data frames for a given duration.
4	BC_REJ	Broadcast frame reject. If asserted, frames with DA (destination address) = FF_FF_FF_FF_FF_FF will be rejected unless the PROM bit is set. If both BC_REJ and PROM = 1, then frames with broadcast DA will be accepted and the M (MISS) bit will be set in the receive buffer descriptor.
3	PROM	Promiscuous mode. All frames are accepted regardless of address matching.
2	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.
1	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).
0	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 $FEC_n\_ETXCLK$ when LOOP is asserted. DRT must be set to zero when asserting LOOP.

### 19.2.4.10 Transmit Control Registers (TCR0 & TCR1)

The  $TCR_n$  is read/write and is written by the user to configure the transmit block. This register is cleared at system reset. Bits 2 and 1 should be modified only when  $ECR_n[ETHER\_EN] = 0$ .

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
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

	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	RFC_PAUSE	TFC_PAUSE	FEDN	HBC	GTS
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Address	IPSBAR + 0x10C4 (FEC0) & IPSBAR + 0x18C4 (FEC1)															

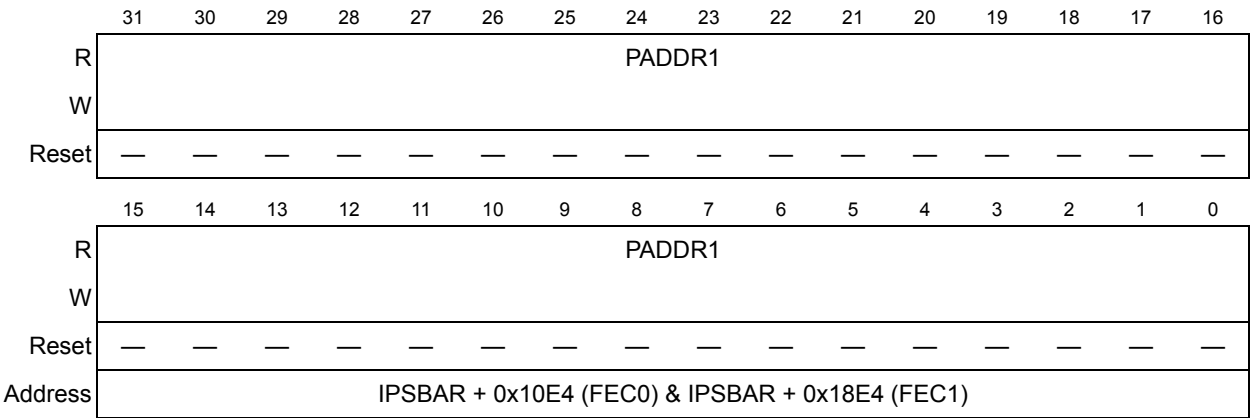
**Figure 19-11. Transmit Control Register (TCR $_n$ )**

**Table 19-14. TCR Field Descriptions**

Bits	Name	Description
31–5	—	Reserved, should be cleared.
4	RFC_PAUSE	Receive frame control pause. This read-only status bit will be 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 will automatically clear when the pause duration is complete.
3	TFC_PAUSE	Transmit frame control pause. Transmits a PAUSE frame when asserted. When this bit is set, the MAC will stop transmission of data frames after the current transmission is complete. At this time, the GRA interrupt in the $EIR_n$ register will be asserted. With transmission of data frames stopped, the MAC will transmit a MAC Control PAUSE frame. Next, the MAC will clear the TFC_PAUSE bit and resume 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.
2	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.
1	HBC	Heartbeat control. If set, the heartbeat check is performed following end of transmission and the HB bit in the status register will be set if the collision input does not assert within the heartbeat window. This bit should only be modified when $ETHER\_EN$ is deasserted.
0	GTS	Graceful transmit stop. When this bit is set, the MAC will stop transmission after any frame that is currently being transmitted is complete and the GRA interrupt in the $EIR_n$ register will be asserted. If frame transmission is not currently underway, the GRA interrupt will be asserted immediately. Once transmission has completed, a “restart” can be accomplished by clearing the GTS bit. The next frame in the transmit FIFO will then be transmitted. If an early collision occurs during transmission when $GTS = 1$ , transmission will stop after the collision. The frame will be transmitted again once GTS is cleared. Note that there may be old frames in the transmit FIFO that will be transmitted when GTS is reasserted. To avoid this deassert $ECR_n[ETHER\_EN]$ following the GRA interrupt.

### 19.2.4.11 Physical Address Low Registers (PALR0 & PALR1)

The PALR $n$  is written by the user. This register contains the lower 32 bits (bytes 0,1,2,3) of the 48-bit 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.



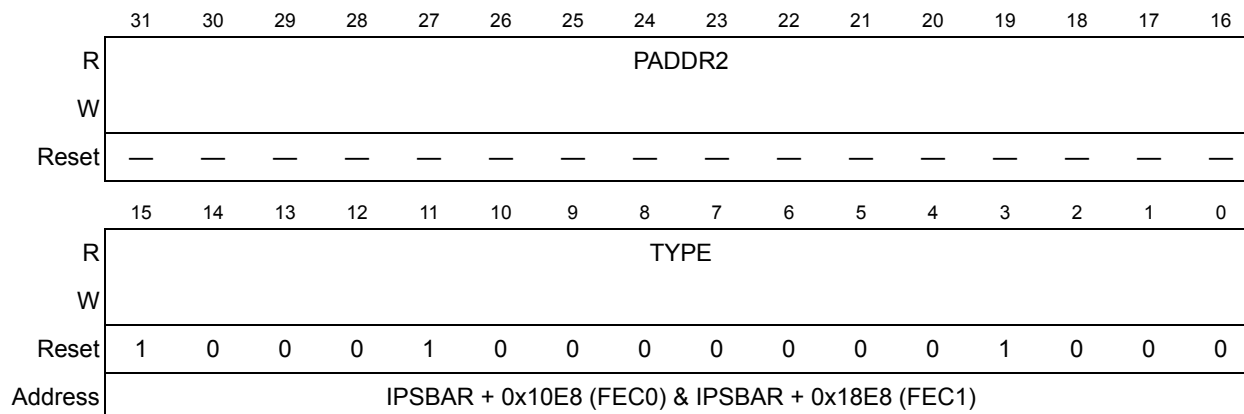
**Figure 19-12. Physical Address Low Register (PALR $n$ )**

**Table 19-15. PALR $n$  Field Descriptions**

Bits	Name	Description
31–0	PADDR1	Bytes 0 (bits 31:24), 1 (bits 23:16), 2 (bits 15:8) and 3 (bits 7:0) of the 6-byte individual address to be used for exact match, and the source address field in PAUSE frames.

### 19.2.4.12 Physical Address High Registers (PAUR0 & PAUR1)

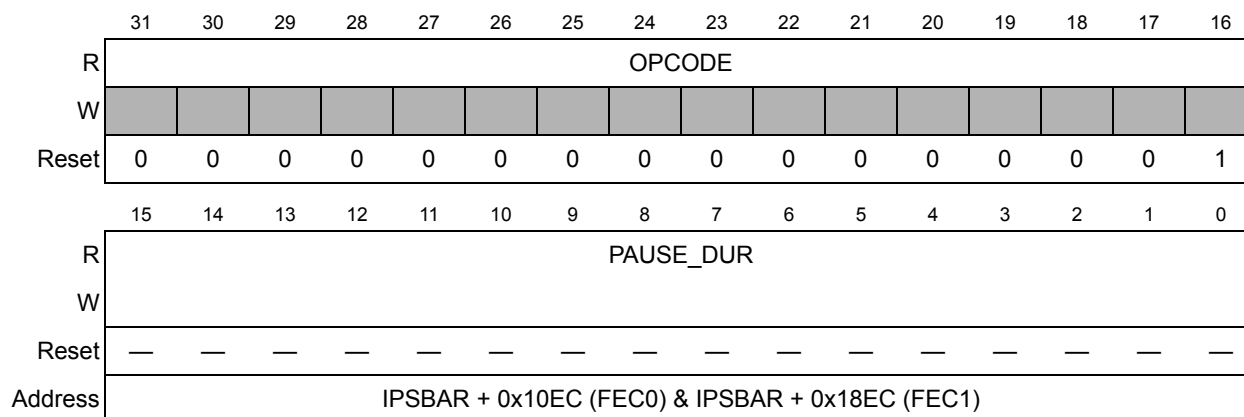
The PAUR $n$  is written by the user. This register contains the upper 16 bits (bytes 4 and 5) of the 48-bit address used in the address recognition process to compare with the DA (destination address) field of receive frames with an individual DA. In addition, this register is used in bytes 4 and 5 of the 6-byte Source Address field when transmitting PAUSE frames. Bits 15:0 of PAUR $n$  contain a constant type field (0x8808) used for transmission of PAUSE frames. This register is not reset and bits 31:16 must be initialized by the user.

Figure 19-13. Physical Address High Register (PAUR<sub>n</sub>)Table 19-16. PAUR<sub>n</sub> Field Descriptions

Bits	Name	Description
31–16	PADDR2	Bytes 4 (bits 31:24) and 5 (bits 23:16) of the 6-byte individual address to be used for exact match, and the Source Address field in PAUSE frames.
15–0	TYPE	Type field in PAUSE frames. These 16-bits are a constant value of 0x8808.

### 19.2.4.13 Opcode/Pause Duration Registers (OPD0 & OPD1)

The OPD<sub>n</sub> is read/write accessible. This register contains the 16-bit opcode and 16-bit pause duration 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 will pause transmission for the duration specified in the pause duration field. This register is not reset and must be initialized by the user.

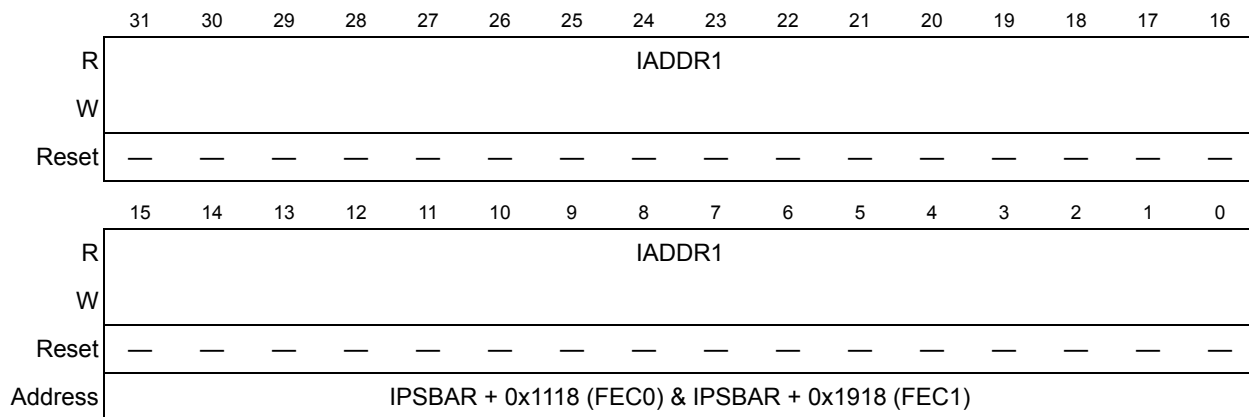
Figure 19-14. Opcode/Pause Duration Register (OPD<sub>n</sub>)

**Table 19-17. OPD<sub>n</sub> Field Descriptions**

Bits	Name	Description
31–16	OPCODE	Opcode field used in PAUSE frames. These bits are a constant, 0x0001.
15–0	PAUSE_DUR	Pause Duration field used in PAUSE frames.

#### 19.2.4.14 Descriptor Individual Upper Address Registers (IAUR0 & IAUR1)

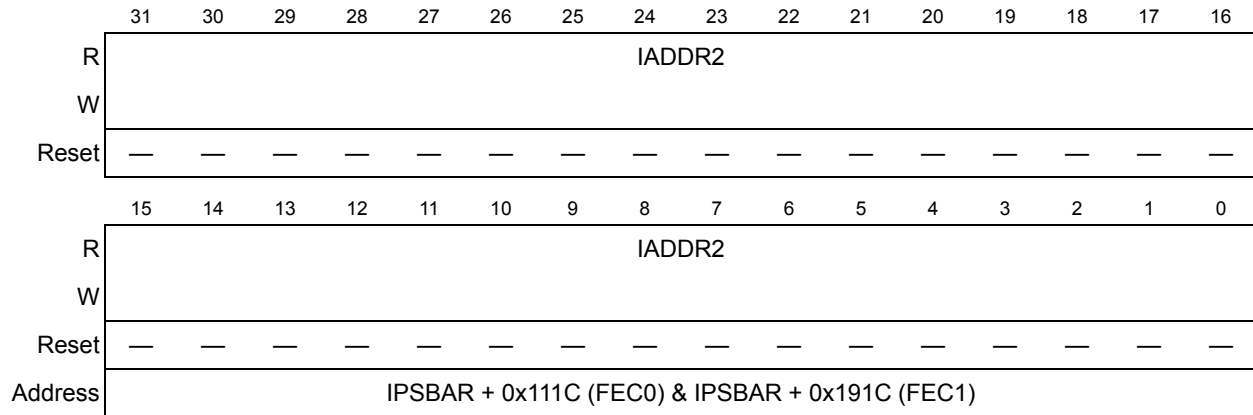
The IAUR<sub>n</sub> is written by the user. This register contains the upper 32 bits of the 64-bit individual address hash table used in the address recognition process to check for possible match with the DA field of receive frames with an individual DA. This register is not reset and must be initialized by the user.

**Figure 19-15. Descriptor Individual Upper Address Register (IAUR<sub>n</sub>)****Table 19-18. IAUR<sub>n</sub> Field Descriptions**

Bits	Name	Descriptions
31–0	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.

#### 19.2.4.15 Descriptor Individual Lower Address Registers (IALR0 & IALR1)

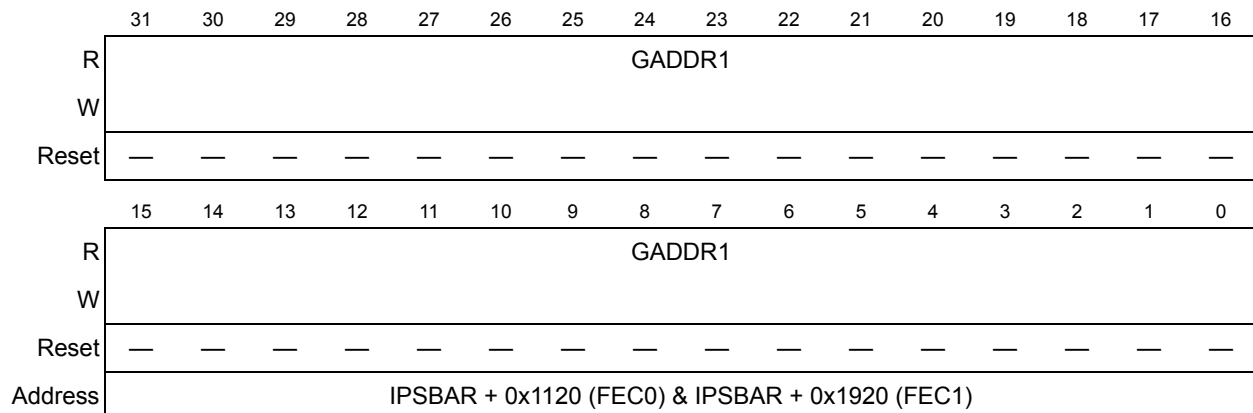
The IALR<sub>n</sub> register is written by the user. This register contains the lower 32 bits of the 64-bit individual address hash table used in the address recognition process to check for possible match with the DA field of receive frames with an individual DA. This register is not reset and must be initialized by the user.

**Figure 19-16. Descriptor Individual Lower Address Register (IALR)****Table 19-19. IALR Field Descriptions**

Bits	Name	Description
31–0	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.

#### 19.2.4.16 Descriptor Group Upper Address Registers (GAUR0 & GAUR1)

The GAUR $n$  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.

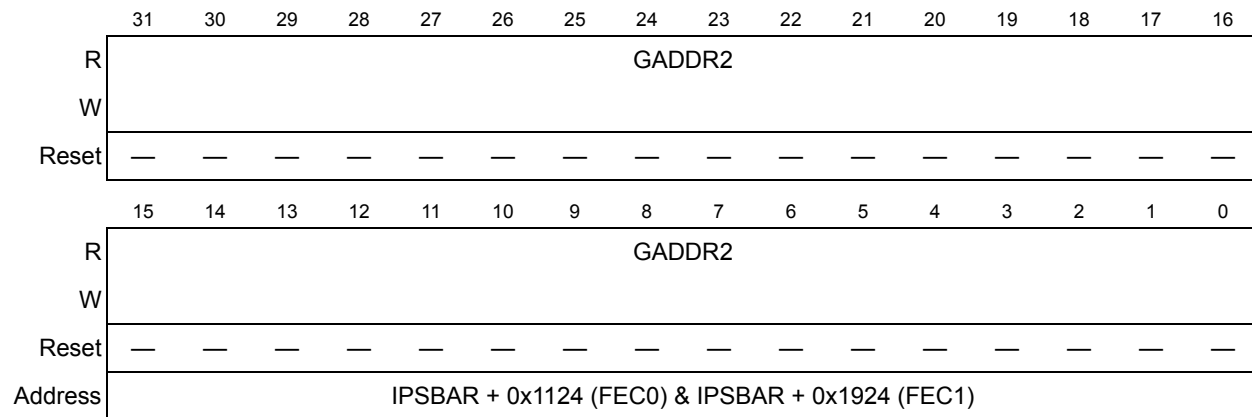
**Figure 19-17. Descriptor Group Upper Address Register (GAUR $n$ )**

**Table 19-20. GAUR Field Descriptions**

Bits	Name	Description
31–0	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.

#### 19.2.4.17 Descriptor Group Lower Address Registers (GALR0 & GALR1)

The  $GALR_n$  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.

**Figure 19-18. Descriptor Group Lower Address Register ( $GALR_n$ )****Table 19-21.  $GALR_n$  Field Descriptions**

Bits	Name	Description
31–0	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.

#### 19.2.4.18 FIFO Transmit FIFO Watermark Registers (TFWR0 & TFWR1)

The  $TFWR_n$  is a 2-bit read/write register 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 will minimize 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).



	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
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
	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	TFWR	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Address	IPSBAR + 0x1144 (FEC0) & IPSBAR + 0x1944 (FEC1)															

Figure 19-19. FIFO Transmit FIFO Watermark Register (TFWR<sub>n</sub>)Table 19-22. TFWR<sub>n</sub> Field Descriptions

Bits	Name	Descriptions
31–2	—	Reserved, should be cleared.
1–0	TFWR	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

#### 19.2.4.19 FIFO Receive Bound Registers (FRBR0 & FRBR1)

The FRBR<sub>n</sub> is an 8-bit register that the user can read to determine the upper address bound of the FIFO RAM. Drivers can use this value, along with the FRSR<sub>n</sub> to appropriately divide the available FIFO RAM between the transmit and receive data paths.

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
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
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
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
Address	IPSBAR + 0x114C (FEC0) & IPSBAR + 0x194C (FEC1)															

Figure 19-20. FIFO Receive Bound Register (FRBR<sub>n</sub>)

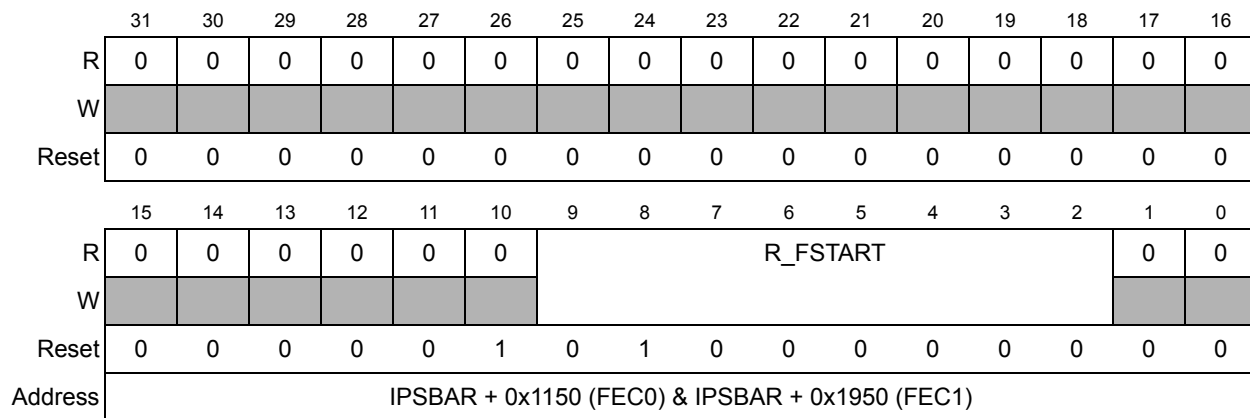
**Table 19-23. FRBR<sub>n</sub> Field Descriptions**

Bits	Name	Descriptions
31–10	—	Reserved, read as 0 (except bit 10, which is read as 1).
9–2	R_BOUND	Read-only. Highest valid FIFO RAM address.
1–0	—	Reserved, should be cleared.

#### 19.2.4.20 FIFO Receive Start Registers (FRSR0 & FRSR1)

The FRSR<sub>n</sub> is programmed by the user to indicate the starting address of the receive FIFO. FRSR<sub>n</sub> 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<sub>n</sub>. The receive FIFO uses addresses from FRSR<sub>n</sub> to FRBR<sub>n</sub> inclusive.

The FRSR<sub>n</sub> register is initialized by hardware at reset. FRSR<sub>n</sub> only needs to be written to change the default value.

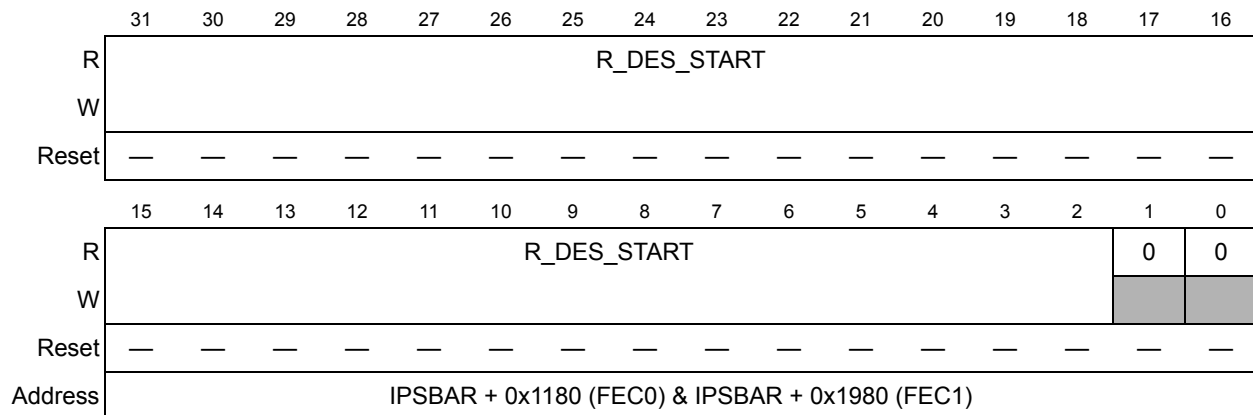
**Figure 19-21. FIFO Receive Start Register (FRSR<sub>n</sub>)****Table 19-24. FRSR<sub>n</sub> Field Descriptions**

Bits	Name	Descriptions
31–10	—	Reserved, read as 0 (except bit 10, which is read as 1).
9–2	R_FSTART	Address of first receive FIFO location. Acts as delimiter between receive and transmit FIFOs.
1–0	—	Reserved, read as 0.

#### 19.2.4.21 Receive Descriptor Ring Start Registers (ERDSR0 & ERDSR1)

The ERDSR<sub>n</sub> 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.



**Figure 19-22. Receive Descriptor Ring Start Register (ERDSR<sub>n</sub>)**

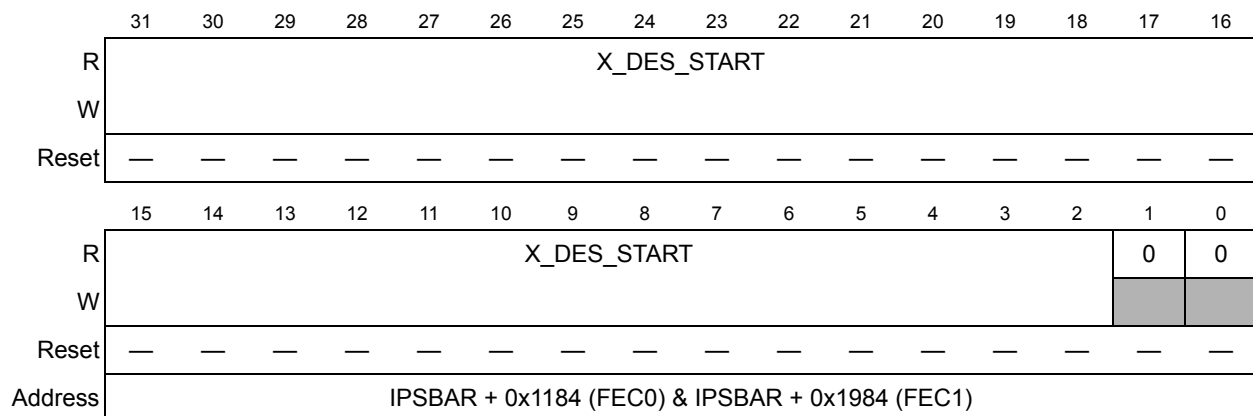
**Table 19-25. ERDSR<sub>n</sub> Field Descriptions**

Bits	Name	Descriptions
31–2	R_DES_START	Pointer to start of receive buffer descriptor queue.
1–0	—	Reserved, should be cleared.

#### 19.2.4.22 Transmit Buffer Descriptor Ring Start Registers (ETSDR0 & ETSDR1)

The ETSDR<sub>n</sub> 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 1 and 0 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.



**Figure 19-23. Transmit Buffer Descriptor Ring Start Register (ETDSR<sub>n</sub>)**

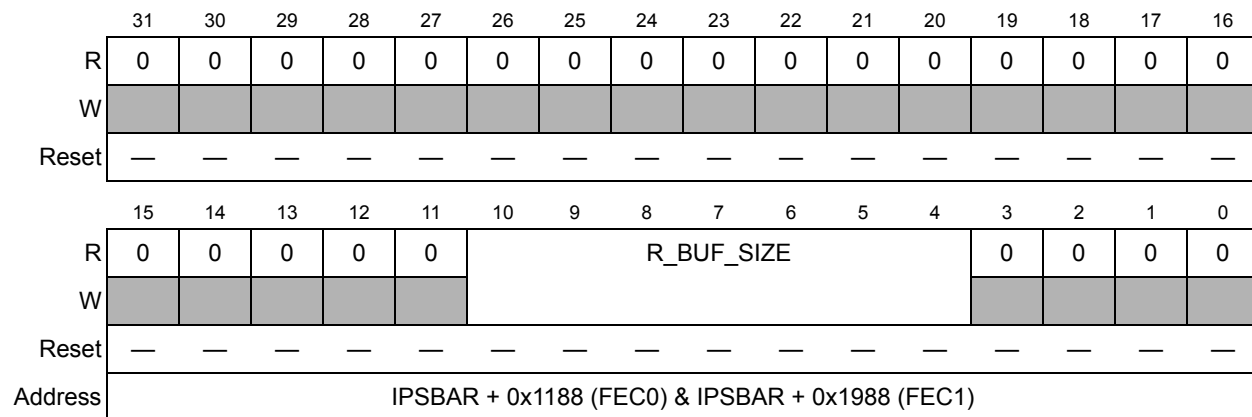
**Table 19-26. ETDSR<sub>n</sub> Field Descriptions**

Bits	Name	Descriptions
31–2	X_DES_START	Pointer to start of transmit buffer descriptor queue.
1–0	—	Reserved, should be cleared.

### 19.2.4.23 Receive Buffer Size Registers (EMRBR0 & EMRBR1)

The EMRBR<sub>n</sub> is a 9-bit register programmed by the user. The EMRBR<sub>n</sub> register dictates the maximum size of all receive buffers. Note that because receive frames will be truncated at 2k-1 bytes, only bits 10–4 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<sub>n</sub> must be set to RCR<sub>n</sub>[MAX\_FL] or larger. The EMRBR<sub>n</sub> must be evenly divisible by 16. To insure this, bits 3-0 are forced low. To minimize bus utilization (descriptor fetches) it is recommended that EMRBR<sub>n</sub> be greater than or equal to 256 bytes.

The EMRBR<sub>n</sub> register does not reset, and must be initialized by the user.

**Figure 19-24. Receive Buffer Size Register (EMRBR<sub>n</sub>)****Table 19-27. EMRBR<sub>n</sub> Field Descriptions**

Bits	Name	Descriptions
30–11	—	Reserved, should be written to 0 by the host processor.
10–4	R_BUF_SIZE	Receive buffer size.
3–0	—	Reserved, should be written to 0 by the host processor.

## 19.2.5 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.

### 19.2.5.1 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 (32-bit aligned 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  $RxBDn[E]$  or  $TxBDn[R]$  bit “produces” the buffer. Software writing to either the  $TDARn$  or  $RDARn$  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  $RxBDn[E]$  or  $TxBDn[R]$  bit will be cleared by hardware to signal 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  $ECRn[ETHER\_EN]$  signal operates as a reset to the BD/DMA logic. When  $ECRn[ETHER\_EN]$  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  $ECRn[ETHER\_EN]$  bit is set.

The buffer descriptors operate as two separate rings.  $ERDSRn$  defines the starting address for receive BDs and  $ETDSRn$  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  $ERDSRn$  and  $ETDSRn$  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.

#### 19.2.5.1.1 Driver/DMA Operation with Transmit BDs

Typically a transmit frame will be divided between multiple buffers. An example is to have an application payload in one buffer, TCP header in a 2nd buffer, IP header in a 3rd buffer, Ethernet/IEEE 802.3 header in a 4th buffer. The Ethernet MAC does not prepend the Ethernet header (destination address, source address, length/type field(s)), 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  $TxBDn[R]$  bits should be set = 1 in reverse order (3rd, 2nd, 1st BD) to insure 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 frame(s) are available by writing to the TDAR $n$  register. When this register is written to (data value is not significant) the FEC RISC will tell the DMA to read the next transmit BD in the ring. Once started, the RISC + DMA will continue 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 will poll this BD one more time. If the R bit = 0 the second time, then the RISC will stop the transmit descriptor read process until software sets up another transmit frame and writes to TDAR $n$ .

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.

#### 19.2.5.1.2 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 $n$  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) will consume 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 will be 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 will be 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 will be 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 2k (2048) bytes or larger are truncated by the FEC at 2032 bytes so software is guaranteed never to see a receive frame larger than 2032 bytes.

Similar to transmit, the FEC will poll the receive descriptor ring after the driver sets up receive BDs and writes to the RDAR $n$  register. As frames are received the FEC will fill receive buffers and update the associated BDs, then read the next BD in the receive descriptor ring. If the FEC

reads a receive BD and finds the E bit = 0, it will poll this BD once more. If the BD = 0 a second time the FEC will stop reading receive BDs until the driver writes to RDAR $n$ .

### 19.2.5.2 Ethernet Receive Buffer Descriptors (RxBD0 & RxBD1)

In the RxBD, the user initializes the E and W bits in the first longword and the pointer in second longword. When the buffer has been DMA'd, the Ethernet controller will modify the E, L, M, BC, MC, LG, NO, CR, OV, and TR bits and write the length of the used portion of the buffer in the first longword. The M, BC, MC, LG, NO, CR, OV and TR bits in the first longword of the buffer descriptor are only modified by the Ethernet controller when the L bit is set.

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Offset + 0	E	RO1	W	RO2	L	—	—	M	BC	MC	LG	NO	—	CR	OV	TR
Offset + 2	Data Length															
Offset + 4	Rx Data Buffer Pointer - A[31:16]															
Offset + 6	Rx Data Buffer Pointer - A[15:0]															

**Figure 19-25. Receive Buffer Descriptor (RxBD $n$ )**

**Table 19-28. Receive Buffer Descriptor Field Definitions**

Word	Bits	Field Name	Description
Offset + 0	15	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	14	RO1	Receive software ownership. This field is reserved for use by software. This read/write bit will not be modified by hardware, nor will its value affect hardware.
Offset + 0	13	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. $n$
Offset + 0	12	RO2	Receive software ownership. This field is reserved for use by software. This read/write bit will not be modified by hardware, nor will its value affect hardware.
Offset + 0	11	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	10–9	—	Reserved.

**Table 19-28. Receive Buffer Descriptor Field Definitions (Continued)**

Word	Bits	Field Name	Description
Offset + 0	8	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	7	BC	Will be set if the DA is broadcast (FF-FF-FF-FF-FF-FF).
Offset + 0	6	MC	Will be set if the DA is multicast and not BC.
Offset + 0	5	LG	Rx frame length violation. Written by the FEC. A frame length greater than RCRn[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 2032 bytes.
Offset + 0	4	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. If this bit is set the CR bit will not be set.
Offset + 0	3	—	Reserved.
Offset + 0	2	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	1	OV	Overflow. 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	0	TR	Will be set if the receive frame is truncated (frame length > 2032 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.
Offset + 2	15–0	Data Length	Data length. Written by the FEC. Data length is the number of octets written by the FEC into this BD's data buffer if L = 0 (the value will be 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.
Offset + 4	15–0	A[31:16]	RX data buffer pointer, bits [31:16] <sup>1</sup>
Offset + 6	15–0	A[15:0]	RX data buffer pointer, bits [15:0]

<sup>1</sup> 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.*n*

### 19.2.5.3 Ethernet Transmit Buffer Descriptors (TxBD0 & TxBD1)

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 TxB<sub>D</sub> the user initializes the R, W, L, and TC bits and the length (in bytes) in the first longword, and the buffer pointer in the second longword.

The FEC will set the R bit = 0 in the first longword 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 19.2.3, “MIB Block Counters Memory Map”](#) for more details.

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Offset + 0	R	TO1	W	TO2	L	TC	ABC	—	—	—	—	—	—	—	—	—
Offset + 2	Data Length															
Offset + 4	Tx Data Buffer Pointer - A[31:16]															
Offset + 6	Tx Data Buffer Pointer - A[15:0]															

**Figure 19-26. Transmit Buffer Descriptor (TxB<sub>D</sub><sub>n</sub>)**

**Table 19-29. Transmit Buffer Descriptor Field Definitions**

Word	Bits	Field Name	Description
Offset + 0	15	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	14	TO1	Transmit software ownership. This field is reserved for software use. This read/write bit will not be modified by hardware, nor will its value affect hardware.
Offset + 0	13	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. <sub>n</sub>
Offset + 0	12	TO2	Transmit software ownership. This field is reserved for use by software. This read/write bit will not be modified by hardware, nor will its value affect hardware.
Offset + 0	11	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	10	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	9	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	8–0	—	Reserved.

**Table 19-29. Transmit Buffer Descriptor Field Definitions (Continued)**

Word	Bits	Field Name	Description
Offset + 2	15–0	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 [15:5] are used by the DMA engine, bits[4:0] are ignored.
Offset + 4	15–0	A[31:16]	Tx data buffer pointer, bits [31:16] <sup>1</sup>
Offset + 6	15–0	A[15:0]	Tx data buffer pointer, bits [15:0].

<sup>1</sup> 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<sub>n</sub> which will trigger the FEC to poll the next BD in the ring.

## 19.3 Functional Description

This section describes the operation of the FECs, 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 FECs.

### 19.3.1 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 FECs.

#### 19.3.1.1 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<sub>n</sub>[ETHER\_EN] bit is cleared. ECR<sub>n</sub>[ETHER\_EN] is deasserted by a hard reset or may be deasserted by software to halt operation. By deasserting ECR<sub>n</sub>[ETHER\_EN], the configuration control registers such as the TCR<sub>n</sub> and RCR<sub>n</sub> will not be reset, but the entire data path will be reset.

**Table 19-30. ECR<sub>n</sub>[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 <sub>n</sub>	Cleared
TDAR <sub>n</sub>	Cleared
Descriptor Controller block	Halt operation

### 19.3.2 User Initialization (Prior to Setting ECR<sub>n</sub>[ETHER\_EN])

The user needs to initialize portions the FEC prior to setting the ECR<sub>n</sub>[ETHER\_EN] bit. The exact values will depend on the particular application. The sequence is not important.

Ethernet MAC registers requiring initialization are defined in [Table 19-31](#).

**Table 19-31. User Initialization (Before ECR<sub>n</sub>[ETHER\_EN])**

Description
Initialize EIMR <sub>n</sub>
Clear EIR <sub>n</sub> (write 0xFFFF_FFFF)
TFWR <sub>n</sub> (optional)
IALR <sub>n</sub> / IAUR <sub>n</sub>
GAUR <sub>n</sub> / GALR <sub>n</sub>
PALR <sub>n</sub> / PAUR <sub>n</sub> (only needed for full duplex flow control)
OPD <sub>n</sub> (only needed for full duplex flow control)
RCR <sub>n</sub>
TCR <sub>n</sub>
MSCR <sub>n</sub> (optional)
Clear MIB_RAM <sub>n</sub> (locations IPSBAR + 0x1200-0x12FC & IPSBAR + 0x2000-0x20FC)

FEC FIFO/DMA registers that require initialization are defined in [Table 19-32](#).

**Table 19-32. FEC User Initialization (Before ECR[ETHER\_EN])**

Description
Initialize FRSR <sub>n</sub> (optional)
Initialize EMRBR <sub>n</sub>
Initialize ERDSR <sub>n</sub>
Initialize ETDSR <sub>n</sub>

**Table 19-32. FEC User Initialization (Before ECR<sub>n</sub>[ETHER\_EN]) (Continued)**

Description
Initialize (Empty) Transmit Descriptor ring
Initialize (Empty) Receive Descriptor ring

### 19.3.3 Microcontroller Initialization

In the FEC, the descriptor control RISC initializes some registers after ECR<sub>n</sub>[ETHER\_EN] is asserted. After the microcontroller initialization sequence is complete, the hardware is ready for operation.

Table 19-33 shows microcontroller initialization operations.

**Table 19-33. 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

### 19.3.4 User Initialization (After Asserting ECR<sub>n</sub>[ETHER\_EN])

After asserting ECR<sub>n</sub>[ETHER\_EN], the user can set up the buffer/frame descriptors and write to the TDAR<sub>n</sub> and RDAR<sub>n</sub>. Refer to [Section 19.2.5, “Buffer Descriptors”](#) for more details.

### 19.3.5 Network Interface Options

The FECs support 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<sub>n</sub>[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 19-34](#) below.

**Table 19-34. MII Mode**

Signal Description	EMAC pin
Transmit Clock	FEC <sub>n</sub> _ETXCLK
Transmit Enable	FEC <sub>n</sub> _ETXEN

**Table 19-34. MII Mode (Continued)**

Signal Description	EMAC pin
Transmit Data	FEC $n$ _ETXD[3:0]
Transmit Error	FEC $n$ _ETXER
Collision	FEC $n$ _ECOL
Carrier Sense	FEC $n$ _ECRS
Receive Clock	FEC $n$ _ERXCLK
Receive Data Valid	FEC $n$ _ERXDV
Receive Data	FEC $n$ _ERXD[3:0]
Receive Error	FEC $n$ _ERXER
Management Data Clock	FEC $n$ _EMDC
Management Data Input/Output	FEC $n$ _EMDIO

The 7-wire serial mode interface (RCR $n$ [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 19-35](#).

**Table 19-35. 7-Wire Mode Configuration**

SIGNAL DESCRIPTION	EMAC PIN
Transmit Clock	FEC $n$ _ETXCLK
Transmit Enable	FEC $n$ _ETXEN
Transmit Data	FEC $n$ _ETXD[0]
Collision	FEC $n$ _ECOL
Receive Clock	FEC $n$ _ERXCLK
Receive Data Valid	FEC $n$ _ERXDV
Receive Data	FEC $n$ _ERXD[0]

### 19.3.6 FEC Frame Transmission

The Ethernet transmitter is designed to work with almost no intervention from software. Once ECR $n$ [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 $n$ ), the MAC transmit logic will assert FEC $n$ \_ETXEN and start 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 $n$ \_ECRS 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 19.3.14.1, “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 will be 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 $n$ .

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 will be asserted, however the entire frame will be transmitted (no truncation).

To pause transmission, set the GTS (graceful transmit stop) bit in the TCR $n$  register. When the TCR $n$ [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 $n$ [GTS] is cleared, the FEC resumes transmission with the next frame.

The Ethernet controller transmits bytes least significant bit first.

### 19.3.7 FEC Frame Reception

The FEC receivers are 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 setting ECR $n$ [ETHER\_EN], it will immediately start processing receive frames. When FEC $n$ \_ERXDV is asserted, the receiver will first check for a valid PA/SFD header. If the PA/SFD is valid, it will be stripped and the frame will be processed by the receiver. If a valid PA/SFD is not found, the frame will be ignored.

In serial mode, the first 16 bit times of RX\_D0 following assertion of FEC<sub>n</sub>\_ERXDV 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 19.3.14.2, “Reception Errors”](#) for more details.

Receive Buffer (RXB) and Frame Interrupts (RFINT) may be generated if enabled by the EIMR<sub>n</sub> 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 will occur and the LG bit in the Receive Buffer Descriptor (RxBD<sub>n</sub>) will be set. See [Section 19.2.5.2, “Ethernet Receive Buffer Descriptors \(RxBD0 & RxBD1\)”](#) for more details.

When the receive frame is complete, the FEC sets the L-bit in the RxBD<sub>n</sub>, writes the other frame status bits into the RxBD<sub>n</sub>, and clears the E-bit. The Ethernet controller next generates a maskable interrupt (RFINT bit in EIR<sub>n</sub>, maskable by RFIEN bit in EIMR<sub>n</sub>), 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.

## 19.3.8 Ethernet Address Recognition

The FECs filter the received frames based on destination address (DA) type — 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 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 19-27](#) illustrates the address recognition

decisions made by the receive block, while [Figure 19-28](#) illustrates the decisions made by the microcontroller.

If the DA is a broadcast address and broadcast reject ( $RCR_n[BC\_REJ]$ ) is deasserted, then the frame will be accepted unconditionally, as shown in [Figure 19-27](#). Otherwise, if the DA is not a broadcast address, then the microcontroller runs the address recognition subroutine, as shown in [Figure 19-28](#).

If the DA is a group (multicast) address and flow control is disabled, then the microcontroller will perform a group hash table lookup using the 64-entry hash table programmed in  $GAUR_n$  and  $GALR_n$ . If a hash match occurs, the receiver accepts the frame.

If flow control is enabled, the microcontroller will do 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 will be rejected. Note the receiver will detect a PAUSE frame with the DA field set to either the designated PAUSE DA or the unicast physical address.

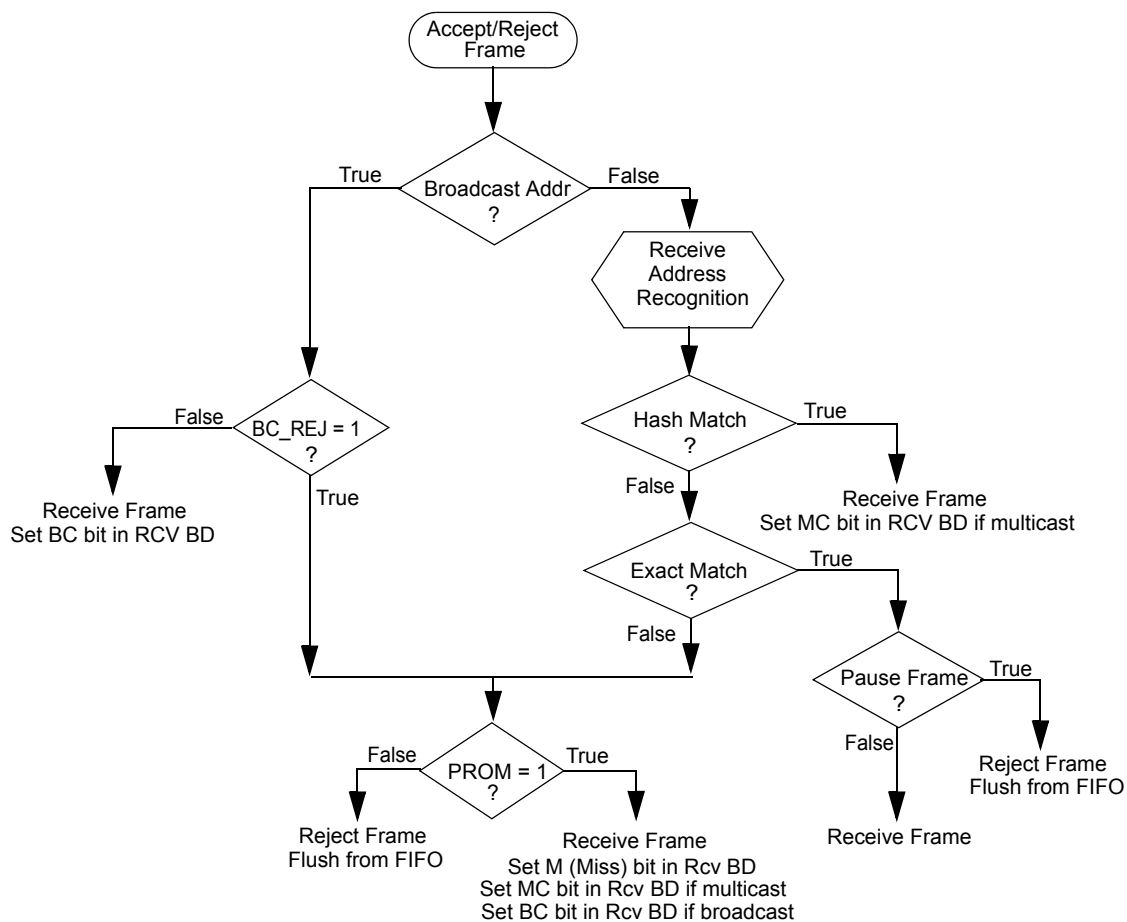
If the DA is the individual (unicast) address, the microcontroller performs an individual exact match comparison between the DA and 48-bit physical address that the user programs in the  $PALR_n$  and  $PAUR_n$  registers. If an exact match occurs, the frame is accepted; otherwise, the microcontroller does an individual hash table lookup using the 64-entry hash table programmed in registers,  $IAUR_n$  and  $IALR_n$ . In the case of an individual hash match, the frame is accepted. Again, the receiver will accept or reject the frame based on PAUSE frame detection, shown in [Figure 19-27](#).

If neither a hash match (group or individual), nor an exact match (group or individual) occur, then if promiscuous mode is enabled ( $RCR_n[PROM] = 1$ ), then the frame will be accepted and the MISS bit in the receive buffer descriptor is set; otherwise, the frame will be rejected.

Similarly, if the DA is a broadcast address, broadcast reject ( $RCR_n[BC\_REJ]$ ) is asserted, and promiscuous mode is enabled, then the frame will be accepted and the MISS bit in the receive buffer descriptor is set; otherwise, the frame will be rejected.

In general, when a frame is rejected, it is flushed from the FIFO.





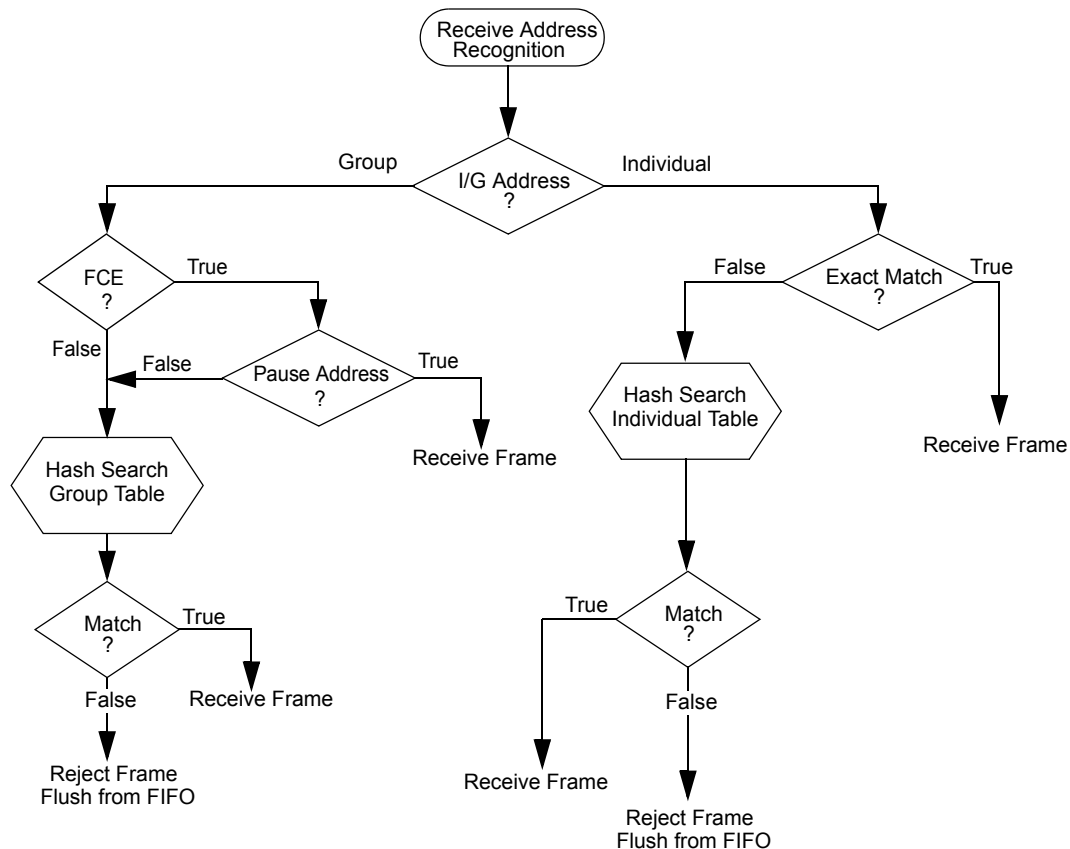
## NOTES:

BC\_REJ - field in RCR register (BroadCast REJect)

PROM - field in RCR register (PROMiscuous mode)

Pause Frame - valid PAUSE frame received

**Figure 19-27. 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 19-28. Ethernet Address Recognition—Microcode Decisions**

### 19.3.9 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  $GAUR_n$ ,  $GALR_n$  (group address hash match) or  $IAUR_n$ ,  $IALR_n$  (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  $GAUR_n$  (msb = 1) or  $GALR_n$  (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 19-36. Destination Address to 6-Bit Hash**

48-bit DA	6-bit Hash (in hex)	Hash Decimal Value
65:ff:ff:ff:ff	0x0	0
55:ff:ff:ff:ff	0x1	1
15:ff:ff:ff:ff	0x2	2
35:ff:ff:ff:ff	0x3	3
b5:ff:ff:ff:ff	0x4	4
95:ff:ff:ff:ff	0x5	5
d5:ff:ff:ff:ff	0x6	6
f5:ff:ff:ff:ff	0x7	7
db:ff:ff:ff:ff	0x8	8
fb:ff:ff:ff:ff	0x9	9
bb:ff:ff:ff:ff	0xa	10
8b:ff:ff:ff:ff	0xb	11
0b:ff:ff:ff:ff	0xc	12
3b:ff:ff:ff:ff	0xd	13
7b:ff:ff:ff:ff	0xe	14
5b:ff:ff:ff:ff	0xf	15
27:ff:ff:ff:ff	0x10	16
07:ff:ff:ff:ff	0x11	17
57:ff:ff:ff:ff	0x12	18
77:ff:ff:ff:ff	0x13	19
f7:ff:ff:ff:ff	0x14	20
c7:ff:ff:ff:ff	0x15	21
97:ff:ff:ff:ff	0x16	22
a7:ff:ff:ff:ff	0x17	23
99:ff:ff:ff:ff	0x18	24
b9:ff:ff:ff:ff	0x19	25

**Table 19-36. Destination Address to 6-Bit Hash (Continued)**

48-bit DA	6-bit Hash (in hex)	Hash Decimal Value
f9:ff:ff:ff:ff:ff	0x1a	26
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

**Table 19-36. Destination Address to 6-Bit Hash (Continued)**

48-bit DA	6-bit Hash (in hex)	Hash Decimal Value
5d:ff:ff:ff:ff:ff	0x3a	58
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

### 19.3.10 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 ( $TCRn[FDEN]$  asserted) and flow control enable ( $RCRn[FCE]$ ) 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 19-37. 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 Duration	0x0000–0xFFFF

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,  $TCRn[GTS]$  is set by the FEC internally. When transmission has paused, the  $EIRn[GRA]$  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  $OPDn[PAUSE\_DUR]$  slot times have expired. On  $OPDn[PAUSE\_DUR]$  expiration,  $TCRn[GTS]$  is deasserted allowing MAC data frame transmission to resume. Note that the receive flow control pause ( $TCRn[RFC\_PAUSE]$ ) status bit is set 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 ( $TCRn[TFC\_PAUSE]$ ). On assertion of transmit flow control pause ( $TCRn[TFC\_PAUSE]$ ), the transmitter sets  $TCRn[GTS]$  internally. When the transmission of data frames stops, the  $EIRn[GRA]$  (graceful stop complete) interrupt asserts. Following  $EIRn[GRA]$  assertion, the pause frame is transmitted. On completion of pause frame transmission, flow control pause ( $TCRn[TFC\_PAUSE]$ ) and  $TCRn[GTS]$  are cleared internally.

The user must specify the desired pause duration in the  $OPDn$  register.

Note that when the transmitter is paused due to receiver/microcontroller pause frame detection, transmit flow control pause ( $TCRn[TFC\_PAUSE]$ ) still may be asserted and will cause the transmission of a single pause frame. In this case, the  $EIRn[GRA]$  interrupt will not be asserted.

### 19.3.11 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 will be ignored and a collision will occur.

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.

### 19.3.12 Collision Handling

If a collision occurs during frame transmission, the Ethernet controller will continue 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 will be 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.

### 19.3.13 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  $RCRn$  register and the FDEN bit in the  $TCRn$  register.

For both internal and external loopback set  $FDEN = 1$ .

For internal loopback set  $RCR_n[LOOP] = 1$  and  $RCR_n[DRT] = 0$ .  $FEC_n\_ETXEN$  and  $FEC_n\_ETXER$  will 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 will cause 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  $RCR_n[LOOP] = 0$ ,  $RCR_n[DRT] = 0$  and configure the external transceiver for loopback.

### 19.3.14 Ethernet Error-Handling Procedure

The Ethernet controller reports frame reception and transmission error conditions using the FEC RxBDs, the  $EIR_n$  register, and the MIB block counters.

#### 19.3.14.1 Transmission Errors

##### 19.3.14.1.1 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  $EIR_n$ . The FEC will then continue to the next transmit buffer descriptor and begin transmitting the next frame.

The “UN” interrupt will be asserted if enabled in the  $EIMR_n$  register.

##### 19.3.14.1.2 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  $EIR_n$ . The FEC will then continue to the next transmit buffer descriptor and begin transmitting the next frame.

The “RL” interrupt will be asserted if enabled in the  $EIMR_n$  register.

##### 19.3.14.1.3 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  $EIR_n$  register. The FEC will then continue to the next transmit buffer descriptor and begin transmitting the next frame.

The “LC” interrupt will be asserted if enabled in the  $EIMR_n$  register.

#### 19.3.14.1.4 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  $TCR_n$  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  $EIR_n$  register, and generates the HBERR interrupt if it is enabled.

### 19.3.14.2 Reception Errors

#### 19.3.14.2.1 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  $RxBD_n$ . All subsequent data in the frame will be discarded and 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.

#### 19.3.14.2.2 Non-Octet Error (Dribbling Bits)

The Ethernet controller handles up to seven 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  $RxBD_n$ . If there is no CRC error, then no error is reported.

#### 19.3.14.2.3 CRC Error

When a CRC error occurs with no dribble bits, the FEC closes the buffer and sets the CR bit in the  $RxBD_n$ . CRC checking cannot be disabled, but the CRC error can be ignored if checking is not required.

#### 19.3.14.2.4 Frame Length Violation

When the receive frame length exceeds  $MAX\_FL$  bytes the BABR interrupt will be generated, and the LG bit in the end of frame  $RxBD_n$  will be set. The frame is not truncated unless the frame length exceeds 2032 bytes).



### **19.3.14.2.5 Truncation**

When the receive frame length exceeds 2032 bytes the frame is truncated and the TR bit is set in the receive BD.



# Chapter 20

## Universal Serial Bus

### 20.1 Introduction

This chapter provides an overview of the universal serial bus (USB) device controller module of the MCF5275. The *USB Specification, Revision 2.0* is a recommended supplement to this chapter. Unless otherwise stated, all mention of the USB specification refers to revision 2.0. It can be downloaded from <http://www.usb.org>. Chapter 2 of this specification, *Terms and Abbreviations*, provides definitions of many of the terms found here.

#### 20.1.1 Block Diagram

A block diagram of the complete USB device is shown in Figure 20-1.

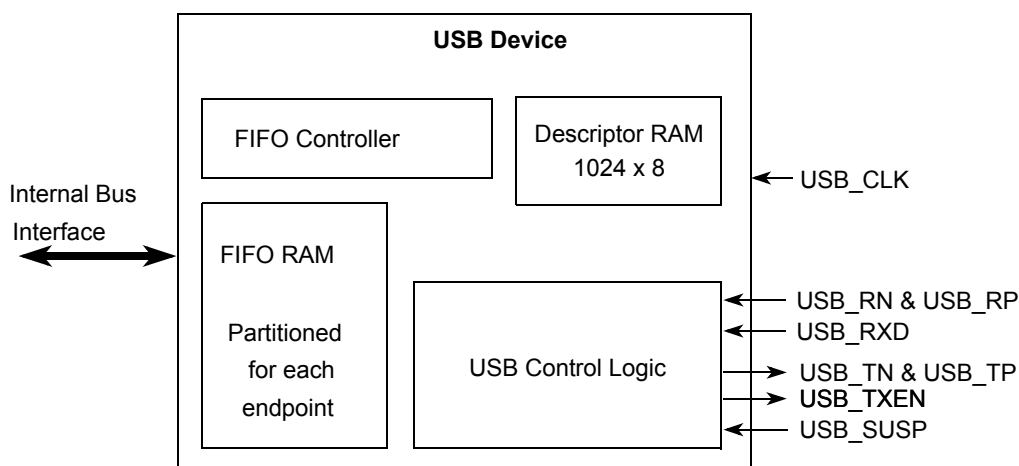


Figure 20-1. USB Device Block Diagram

#### 20.1.2 Overview

The universal serial bus specification describes three types of devices that can connect to the bus. The USB host is the bus master, and periodically polls peripherals to initiate data transfers. There is only one host on the bus. The USB function (or USB device) is a bus slave. It can communicate only with a USB host. It does not generate bus traffic and only responds to requests from the host. A USB hub is a special class of USB function that adds additional connection points to the bus for more USB devices.

The USB device operates under the control of the CPU through the internal peripheral bus. From the user's perspective, this device hides all direct interaction with the USB protocol. The registers

allow the user to enable or disable the device, control characteristics of individual endpoints, and monitor traffic flow through the device without ever seeing the low-level details of the USB protocol.

While this device hides all direct interaction with the protocol, some knowledge of the USB is required in order to properly configure the device for operation on the bus. This document covers programming requirements. Additional information may be found in the USB specification.

### NOTE

The GPIO module must be configured to enable the peripheral function of the appropriate pins (refer to [Chapter 12, “General Purpose I/O Module”](#)) prior to configuring the USB module.

## 20.1.3 Features

The USB device provides the following USB features to the user:

- Supports full speed 12-Mbps and low speed 1.5-Mbps USB operation
- Compliant with *Universal Serial Bus Specification, Revision 2.0*
- Supports up to four endpoints. Endpoint 0 is required by the USB specification, but all others are optional. Endpoint configurations as shown in [Table 20-1](#).

**Table 20-1. Endpoint Configurations**

Endpoint	Type	FIFO Size	Data Transfer	Comments
0	IN	Variable	Control	Mandatory
	OUT	Variable		Mandatory
1-3	IN or OUT	Variable	Ctrl, Int, Bulk, or Iso	Optional

- USB descriptors are stored in a dedicated 1-Kbyte descriptor RAM. The RAM is accessible from the internal peripheral bus via registers within this device.
- One FIFO RAM per USB endpoint. FIFO controllers share a 2-Kbyte RAM made of four contiguous 512-byte RAM's. User programmable registers allow on-the-fly configuration of individual FIFO sizes.
- Isochronous communications pipes are supported. A frame match interrupt feature is supported in order to notify the user when a specific USB frame occurs.
- Remote wake-up feature is supported via a register bit.
- Self powered

## 20.1.4 Modes of Operation

This device provides two modes of operation to the user:

- USB disabled: In this mode, the USB device's datapath will not accept transactions received on the USB. The user has read/write access to the descriptor RAM.
- USB enabled: In this mode, the USB device's datapath is enabled to accept transactions received on the USB. The user does not have read/write access to the descriptor RAM.

## 20.2 External Signal Description

The external interface of the device consists of the signals that connect off-chip. The external interface is described in [Table 20-2](#).

**Table 20-2. USB Signals**

Name	Direction	Function	Reset State <sup>1</sup>
USB_CLK	I	48 Mhz <sup>2</sup> (or 6 Mhz) clock used to derive 12 MHz (or 1.5 MHz) USB bit clock.	—
USB_SPEED	O	Signals between full or low speed.	Asserted
USB_RN	I	Half of differential received USB signal.	—
USB_RP	I	Half of differential received USB signal.	—
USB_RXD	I	Input data from differential input receiver.	—
USB_TN	O	Half of differential output.	Tri-state
USB_TP	O	Half of differential output.	Tri-state
USB_TXEN	O	Output enable.	Deasserted
USB_SUSP	O	Force transceiver into low power state.	Deasserted

<sup>1</sup> The default reset state of these pins is GPIO. See [Chapter 12, "General Purpose I/O Module"](#) for more information.

<sup>2</sup> Specific clock requirements are set forth in the *Universal Serial Bus Specification, Revision 2.0*, Chapter 7 "Electrical Specification."

### 20.2.1 USB Clock (USB\_CLK)

The 48 Mhz clock is used by the USB device for both clock recovery and generation of a 12 Mhz internal bit clock.

### 20.2.2 USB Speed (USB\_SPEED)

Applications which make use of low speed USB signalling must be able to switch the USB transceiver between low speed and full speed operations. Software has control of this function by driving the state of the USB\_SPD bit in the USB\_CR register onto the USB\_SPEED pin.

### 20.2.3 USB Negative/Positive Receive Signal Inputs (USB\_RN & USB\_RP)

These signals are each one half of the differential USB signal, and is extracted from the USB cable via a single ended input buffer on the external PHY. This signal is used by the module for detecting the single ended 0 (SE0) USB bus state.

### 20.2.4 USB Receive Data (USB\_RXD)

Input data from the differential input receiver. USB\_RXD is the single-ended data extracted from the USB\_RP and USB\_RN signals via a differential input buffer.

### 20.2.5 USB Suspended (USB\_SUSP)

After a long period of inactivity (6.0 ms minimum), the USB will enter suspend mode, indicated on the interface by an active state on USB\_SUSP. During this mode, the device is supposed to enter a low power state while waiting for a wake-up from the USB Host. When the device enters suspend mode, it asserts the suspend signal which forces the PHY into a low power state. When the device leaves suspend mode, USB\_SUSP is deasserted, enabling the PHY for normal USB operations.

### 20.2.6 USB Negative/Positive Transmit Signal Output (USB\_TN & USB\_TP)

These signals are each one half of the differential NRZI formatted output from the USB module. It is fed to the transmitted D- and D+ input of the external PHY.

### 20.2.7 USB Transmit Enable (USB\_TXEN)

This signal is an active low output enable for the differential drivers on the PHY. When this signal is active, the differential drivers will drive the USB. When this signal is inactive, the differential drivers will tristate their outputs.

## 20.3 Memory Map/Register Definition

This section provides a detailed description of all memory and registers accessible to the end user.

The names, addresses, and brief descriptions are shown in [Table 20-3](#). All registers are 32 bits wide and 32 bit aligned. The user may access on a byte, word, or longword basis.

**Table 20-3. Memory Map**

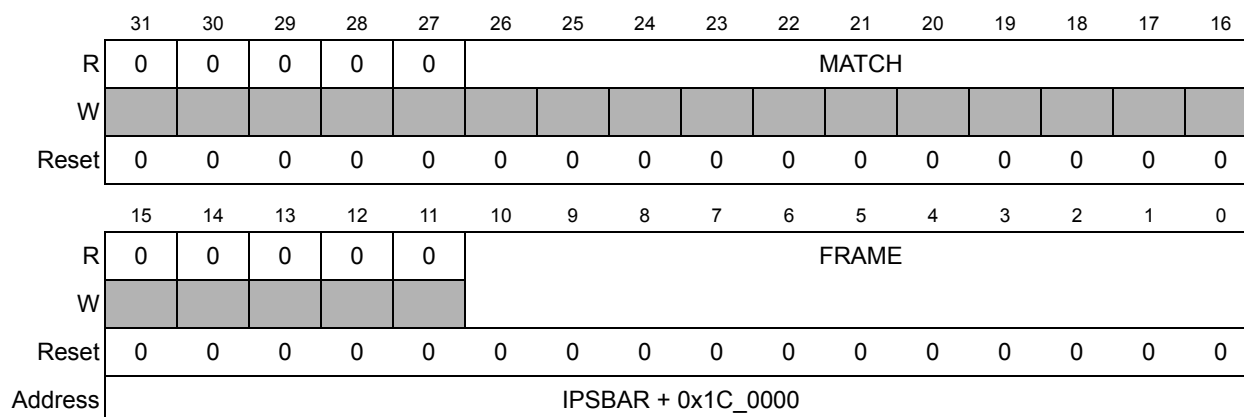
IPSBAR Offset	Mnemonic	[31:24]	[23:16]	[15:8]	[7:0]
0x1C_0000	USB_FRAME	USB Frame Number and Match			
0x1C_0004	USB_SPEC	USB Specification/Release Number			
0x1C_0008	USB_SR	USB Status Register			
0x1C_000C	USB_CR	USB Control Register			
0x1C_0010	USB_DAR	USB Descriptor RAM Address			
0x1C_0014	USB_DDR	USB Descriptor RAM/Endpoint Buffer Data			
0x1C_0018	USB_ISR	USB Interrupt Status Register			
0x1C_001C	USB_IMR	USB Interrupt Mask Register			
0x1C_0020	USB_MCR	USB FIFO Memory Control			
0x1C_0024–0x1C_002C	—	Reserved			
0x1C_0030+( $n \times 0x30$ ) <sup>1</sup>	USB_EP $n$ SR	Endpoint $n$ Status/Control			
0x1C_0034+( $n \times 0x30$ )	USB_EP $n$ ISR	Endpoint $n$ Interrupt Status			
0x1C_0038+( $n \times 0x30$ )	USB_EP $n$ IMR	Endpoint $n$ Interrupt Mask			
0x1C_003C+( $n \times 0x30$ )	USB_EP $n$ FDR	Endpoint $n$ FIFO Data			
0x1C_0040+( $n \times 0x30$ )	USB_EP $n$ FSR	Endpoint $n$ FIFO Status			
0x1C_0044+( $n \times 0x30$ )	USB_EP $n$ FCR	Endpoint $n$ FIFO Control			
0x1C_0048+( $n \times 0x30$ )	USB_EP $n$ LRFP	Endpoint $n$ FIFO Last Read Frame Pointer			
0x1C_004C+( $n \times 0x30$ )	USB_EP $n$ LWFP	Endpoint $n$ FIFO Last Write Frame Pointer			
0x1C_0050+( $n \times 0x30$ )	USB_EP $n$ FAR	Endpoint $n$ FIFO Alarm			
0x1C_0054+( $n \times 0x30$ )	USB_EP $n$ FRP	Endpoint $n$ FIFO Read Pointer			
0x1C_0058+( $n \times 0x30$ )	USB_EP $n$ FWP	Endpoint $n$ FIFO Write Pointer			
0x1C_005C+( $n \times 0x30$ )	—	Reserved			

<sup>1</sup> 'n' refers to the number of endpoints: 0, 1, 2, and 3.

### 20.3.1 Register Descriptions

This section gives detailed descriptions of both the user accessible registers and the bits within the USB device.

### 20.3.1.1 USB Frame Number and Match Register (USB\_FRAME)

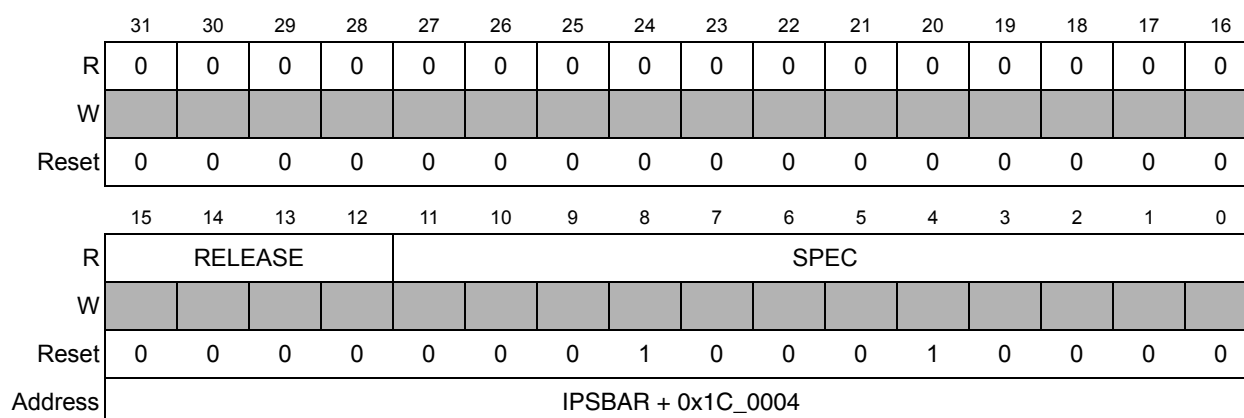


**Figure 20-2. USB Frame Number and Match Register (USB\_FRAME)**

**Table 20-4. USB\_FRAME Field Descriptions**

Bits	Name	Description
31–27	—	Reserved, should be cleared.
26–16	MATCH	When the value in FRAME equals the value in MATCH, then an interrupt will be generated (if not masked).
15–11	—	Reserved, should be cleared.
10–0	FRAME	This is the 11 bit frame number decoded from the SOF packet which leads off each USB frame.

### 20.3.1.2 USB Specification/Release Number Register (USB\_SPEC)



**Figure 20-3. USB Specification/Release Number Register (USB\_SPEC)**



**Table 20-5. USB\_SPEC Field Descriptions**

Bits	Name	Description
31-16	—	Reserved, should be cleared.
15-12	RELEASE	This is the silicon release number of the USB device.
11-0	SPEC	This is the version of USB spec to which the underlying USB core is compliant. 0x100 = version 1.0, 0x110 = version 1.1, 0x200 = version 2.0

### 20.3.1.3 USB Status Register (USB\_SR)

The read-only USB status register reports the current state of various features of the USB device.

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
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

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	RST	SUSP	CFG		INTF		ALTSET		
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Address	IPSBAR + 0x1C_0008															

**Figure 20-4. USB Status Register (USB\_SR)****Table 20-6. USB\_SR Field Descriptions**

Bits	Name	Description
31-9	—	Reserved, should be cleared.
8	RST	Indicates whether the USB is in reset or normal signalling 0 Normal signalling in progress. 1 Reset signalling in progress.
7	SUSP	Suspend status. 0 USB is not suspended. 1 USB is suspended.
6-5	CFG	Indicates the currently selected USB configuration.
4-3	INTF	Indicates which USB interface in the current configuration that ALTSET is associated with.
2-0	ALTSET	Indicates the currently selected USB alternate interface.

### 20.3.1.4 USB Control Register (USB\_CR)

The USB control register configures features of the USB device.

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
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
	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	CO	CE	SPD	EN	RST	0	RES
W																UME
Reset	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
Address	IPSBAR + 0x1C_000C															

Figure 20-5. USB Status Register (USB\_CR)

Table 20-7. USB\_CR Field Descriptions

Bits	Name	Description												
31–7	—	Reserved, should be cleared.												
6	CO	Command Over. Indicates that processing of a device request command has completed. See table in CE description for usage. Automatically clears after the USB core has completed the status phase of a control transfer.												
5	CE	Command Error. Indicates that an error has been encountered during processing of a device request. CO and CE combine to create the handshake code for the status phase of a device request transaction. <table border="1"> <tr> <th>CO</th><th>CE</th><th>Result of Transfer</th></tr> <tr> <td>0</td><td>X</td><td>Application is busy completing the request</td></tr> <tr> <td>1</td><td>0</td><td>Application processes device request successfully</td></tr> <tr> <td>1</td><td>1</td><td>Application encountered an error processing the request</td></tr> </table>	CO	CE	Result of Transfer	0	X	Application is busy completing the request	1	0	Application processes device request successfully	1	1	Application encountered an error processing the request
CO	CE	Result of Transfer												
0	X	Application is busy completing the request												
1	0	Application processes device request successfully												
1	1	Application encountered an error processing the request												
4	SPD	USB Speed. Sets low or full speed operation for the USB module. 0 Low Speed 1 Full Speed  WARNING: The USBSPD bit must be set to match the clock on the USB device (48 Mhz clock for full speed and 6Mhz clock for low speed). Once set for a particular application, do not change or unpredictable operation of the USB device will result.												
3	EN	Enable. Determines if the USB module will respond to requests from the USB host. The device exits reset in the disabled state. The user must ensure that the USB endpoint configuration, descriptor tables, and USB registers are programmed appropriately before enabling communications. This bit does not affect the underlying USB core, only the PHY's ability to communicate with the core. 0 USB module is disabled. All transactions to or from the USB device will be ignored. 1 USB module is enabled and ready to communicate with the host.												
2	RST	Reset. Executes a hard reset sequence on the USB device. This bit allows the system software to force a reset of the USB device's logic when the system is first connected to the USB, or for debug purposes.												

**Table 20-7. USB\_CR Field Descriptions (Continued)**

Bits	Name	Description
1	—	Reserved, should be cleared.
0	RESUME	Initiates resume signalling on the USB. If remote wake-up capability is enabled for the current USB configuration, setting this bit will cause the USB module to initiate resume signalling on the bus. This bit automatically resets to 0 after a write of 1. Writing a 0 to this bit has no effect. If remote wake-up capability has been disabled in the USB device via the CLEAR_FEATURE request, then this bit will have no effect. Any user software should have a time-out feature which aborts the remote wake-up attempt after some time if the RESUME interrupt does not occur.

### 20.3.1.5 USB Descriptor RAM Address Register (USB\_DAR)

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	CFG	BSY	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	1	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	DADR									
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Address	IPSBAR + 0x1C_0010															

**Figure 20-6. USB Descriptor RAM Address Register (USB\_DAR)****Table 20-8. USB\_DAR Field Descriptions**

Bits	Name	Description
31	CFG	Read only. Configuration vs. Descriptor access indicator. This bit is set automatically at power-on or hard reset and clears after the last byte of endpoint buffer configuration data has been downloaded into the USB device. 0 The USB_DDR register is currently set to access the descriptor storage RAM. 1 The USB_DDR register is currently set to download endpoint buffer configuration data to the USB device.
30	BSY	Read only. Configuration download interface busy signal. Because the external PHY and the USB device operate on different clocks, a busy indicator is provided to ensure that writes from the USB_DDR register have a sufficient amount of time to successfully enter the USB device's clock domain. 0 No write is in progress. 1 A write is in progress to the USB device's endpoint buffer. Attempt no other operations on the USB device until BSY has cleared.

**Table 20-8. USB\_DAR Field Descriptions (Continued)**

Bits	Name	Description
29-10	—	Reserved, should be cleared.
9-0	DADR	Descriptor RAM address. Allows user access to the USB descriptor RAM. The user programs a desired address into the DADR bits, then follows it with a read or write to the USB_DDR register to complete the access. This 10-bit address is an offset address from the base of the USB descriptor RAM. Upon read/write access to USB_DDR, the address in DADR will increment automatically. If the CFG bit is set to 1, then DADR is ignored.

### 20.3.1.6 USB Descriptor RAM/Endpoint Buffer Data Register (USB\_DDR)

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
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
	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	DDAT							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Address	IPSBAR + 0x1C_0014															

**Figure 20-7. USB Descriptor RAM/Endpoint Buffer Data (USB\_DDR)****Table 20-9. USB\_DDR Field Descriptions**

Bits	Name	Description
31-8	—	Reserved, should be cleared.
7-0	DDAT	Endpoint buffer/descriptor RAM data register. Allows user access to the USB descriptor memory or to the endpoint buffer download facility. For descriptor access, software programs address into the DADR[10:0] bits and follow with a read or write to the USB_DDR register to complete the access. Upon the read/write access, the address in USB_DAR will increment automatically. For endpoint buffer access, writes to this register cause the data to be loaded into the USB device's endpoint buffers. Access to this register is only allowed when the USBENA bit in the USB_CR register is set to 0. Access at other times is ignored and reads are undefined. When the CFG bit in USB_DAR is set, writes to USB_DDR register go to the USB device endpoint/config buffers, and reads are undefined. When the CFG bit in USB_DAR is clear, writes to USB_DDR go to the descriptor RAM, and reads come from the descriptor RAM.

### 20.3.1.7 USB Interrupt Status Register (USB\_ISR)

The USB interrupt status register maintains the status of interrupt conditions pertaining to USB functions. An interrupt, once set, remains set until cleared by writing a 1 to the corresponding bit. Interrupts do not clear automatically if the event that caused them goes away (for example, if reset

signalling comes and goes with no intervention from software, both RESET\_START and RESET\_STOP would be set). Writing a 0 has no effect.

If a register write occurs at the same time an interrupt is received, the interrupt takes precedence over the write.

Refer to [Section 20.6.2.1, “USB Global Interrupt”](#) for more information about these interrupt types.

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
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
	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	MSOF	SOF	RSTP	RSTR	RES	SUSP	FM	CC
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Address	IPSBAR + 0x1C_0018															

**Figure 20-8. USB Interrupt Status Register (USB\_ISR)**

**Table 20-10. USB\_ISR Field Descriptions**

Bits	Name	Description
31–8	—	Reserved, should be cleared.
7	MSOF	Missed start of frame interrupt. 0 No missed start of frame. 1 An SOF interrupt was set, but not cleared before the next one was received.
6	SOF	USB start of frame interrupt. 0 No start of frame received 1 A start of frame token has been received by the USB device. The USB frame number is current.
5	RSTP	Indicates the end of reset signalling on the USB device. 0 Reset signalling has not stopped. Does not imply that reset signalling is occurring, just that no end-of-reset event has occurred. 1 Reset signalling has stopped.
4	RSTR	Indicates start of reset signalling on the USB device. 0 No reset in progress. 1 USB Reset in progress.
3	RES	Indicates a state change from suspend to resume in the USB device. This bit only indicates the change from suspended to active mode. Clearing the interrupt has no effect on the actual state of the USB. 0 Indicates that USB has not left the suspended state (but does not imply that the bus is, or ever was suspended). 1 USB has left suspend state.

**Table 20-10. USB\_ISR Field Descriptions (Continued)**

Bits	Name	Description
2	SUSP	Indicates a suspend state in the USB device. This bit only indicates the change from active to suspended mode. Clearing the interrupt has no effect on the actual state of the USB. 0 USB is not suspended, or the interrupt was cleared. 1 USB is suspended.
1	FM	Indicates a match between the USB frame number and the frame match register. 0 No match. 1 Match occurred.
0	CC	This indicates that a USB configuration (configuration, interface, alternate) change occurred and the software will need to reread the USB Status Register. 0 No activity. 1 Configuration change occurred.

### 20.3.1.8 USB Interrupt Mask Register (USB\_IMR)

This is the USB interrupt mask register. Setting a bit in this register masks the corresponding interrupt in the USB\_ISR register. Upon reset, all interrupts are masked.

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
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

	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	MSOF	SOF	RSTP	RSTR	RES	SUSP	FM	CC
W																
Reset	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
Address	IPSBAR + 0x1C_001C															

**Figure 20-9. USB Interrupt Mask Register (USB\_IMR)**

### 20.3.1.9 USB FIFO Memory Control Register (USB\_MCR)

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	MEMC		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
Address	IPSBAR + 0x1C_0020															

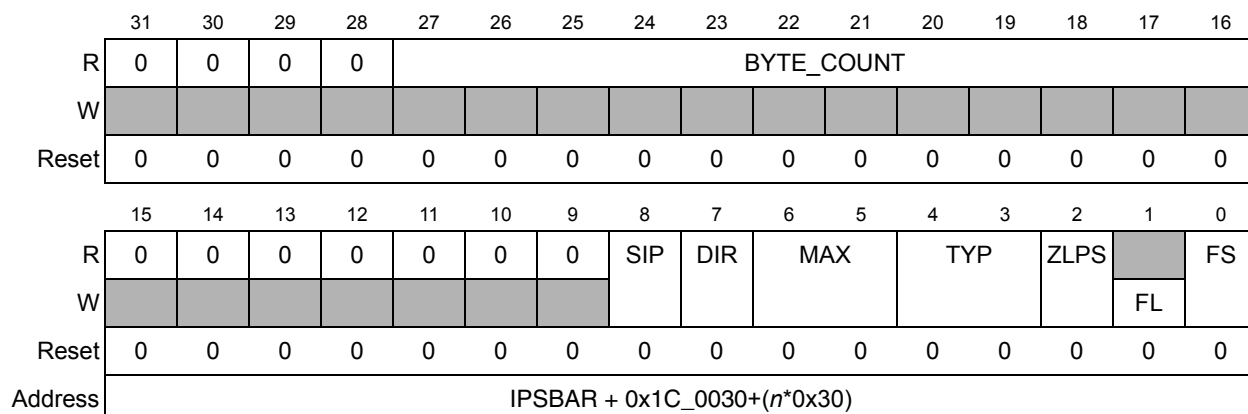
**Figure 20-10. USB Memory Control Register (USB\_MCR)**

**Table 20-11. USB\_MCR Field Descriptions**

Bits	Name	Description																									
31–26	—	Reserved, should be cleared.																									
25–24	MEMC	<div>Selects a size for the FIFOs in the USB device. The FIFO RAM configurations are shown below.<table><tr><th>MEMC[1:0]</th><th>FIFO 0</th><th>FIFO 1</th><th>FIFO 2</th><th>FIFO 3</th></tr><tr><td>00</td><td>32</td><td>2016</td><td>0</td><td>0</td></tr><tr><td>01</td><td>32</td><td>1024</td><td>992</td><td>0</td></tr><tr><td>10</td><td>32</td><td>992</td><td>512</td><td>512</td></tr><tr><td>11</td><td>32</td><td>992</td><td>768</td><td>256</td></tr></table></div>	MEMC[1:0]	FIFO 0	FIFO 1	FIFO 2	FIFO 3	00	32	2016	0	0	01	32	1024	992	0	10	32	992	512	512	11	32	992	768	256
MEMC[1:0]	FIFO 0	FIFO 1	FIFO 2	FIFO 3																							
00	32	2016	0	0																							
01	32	1024	992	0																							
10	32	992	512	512																							
11	32	992	768	256																							
23–0	—	Reserved, should be cleared.																									

### 20.3.1.10 Endpoint *n* Status/Control Register (USB\_EP*n*SR)

The control register allows the user to configure specific aspects of an individual endpoint.

Figure 20-11. USB Endpoint *n* Status/Control Register (USB\_EP*n*SR)Table 20-12. USB\_EP*n*SR Field Descriptions

Bits	Name	Description
31–28	—	Reserved, should be cleared
27–16	BYTE_COUNT	Indicates the number of bytes currently stored in the associated FIFO. The width of this read-only field is dependent on the size of the FIFO. The low order bit of this field is always at bit position sixteen. The high order bit is dependent on the size of the FIFO. Bit positions above the FIFO size are treated as undefined and are unaffected by read or write operations. The maximum field width is twelve bits.
15–9	—	Reserved, should be cleared
8	SIP	Setup packet in progress indicator. 0 Setup data is not being transferred from host to device. 1 Setup data currently being transferred from host to device.
7	DIR	Transfer direction, ignored for control endpoints. 0 OUT Endpoint (from host to device). 1 IN Endpoint (from device to host).
6–5	MAX	Maximum packet size for the endpoint, ignored for isochronous endpoints. 00 8 bytes 01 16 bytes 10 32 bytes 11 64 bytes
4–3	TYP	Indicates the endpoint type. 00 Control 01 Isochronous 10 Bulk 11 Interrupt



**Table 20-12. USB\_EPnSR Field Descriptions (Continued)**

Bits	Name	Description
2	ZLPS	Zero length packet send signal. 0 If the FIFO is empty and the host requests a transaction, the USB device will respond with a NAK. 1 If the FIFO is empty, and the USB host requests an IN transaction, the USB device will send a zero length packet in response. This bit automatically clears once the transaction completes successfully and is used to signal the USB host that the end of data has been reached in a data transmission when the end of data lands on a packet boundary. In that case, there is no short packet to signal the end of data. This bit allows the software to signal that an empty packet should go back to the host.
1	FL	Flushes the associated FIFO. This bit is write-only. 0 Do nothing. 1 Initiate flush operation.
0	FS	When written, causes the endpoint to respond with a STALL to the next poll. This bit automatically clears when the stall takes effect. 0 Do nothing. 1 Force a stall condition. <b>Note:</b> There is no endpoint stalled indicator. None is returned from the USB device. The USB host is expected to communicate with the USB device via device requests to fix the stall condition. The USB host will then send a CLEAR_FEATURE request to the USB device to clear the stall and resume normal operations.

### 20.3.1.11 Endpoint *n* Interrupt Status Register (USB\_EPnISR)

The endpoint interrupt status register monitors the status of a specific endpoint and generates a CPU interrupt each time a monitored event occurs.

An interrupt, once set, remains set until cleared by writing a 1 to the corresponding bit. Interrupts do not clear automatically if the event that caused them goes away (for example, if reset signalling comes and goes with no intervention from software, both RESET\_START and RESET\_STOP would be set). Writing a 0 has no effect.

If a register write occurs at the same time an interrupt is received, the interrupt takes precedence over the write.

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
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
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	FF	FM	FE	FH	FL	MDR	EOT	DR	EOF
W																
Reset	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
Address	IPSBAR + 0x1C_0034+(n*0x30)															

**Figure 20-12. USB Endpoint *n* Interrupt Status Register (USB\_EP*n*ISR)****Table 20-13. USB\_EP*n*ISR Field Descriptions**

Bits	Name	Description
31–9	—	Reserved, should be cleared.
8	FF	FIFO full. 0 FIFO is not full. 1 FIFO is full.
7	FM	FIFO empty. 0 FIFO is not empty. 1 FIFO is empty.
6	FE	Indicates an error condition in the FIFO controller. The error condition can be checked by reading the FIFO status register (USB_EP <i>n</i> FSR). 0 No Error condition pending. 1 Error condition pending.
5	FH	Indicates that the number of bytes in the FIFO has surpassed the high level alarm value.
4	FL	Indicates that the number of bytes in the FIFO has fallen below the FIFO low level alarm value.
3	MDR	Asserts when a device request (DR) interrupt is pending and another setup packet is received. Will only assert for control endpoints. 0 Multiple setup packets are not pending. 1 Multiple setup packets pending.
2	EOT	Indicates that the last packet of a USB data transfer has crossed into, or out of, the USB device. Any packet shorter than the maximum packet size for the associated endpoint is considered to be an end of transfer marker. In addition, for control endpoints only, the EOT interrupt will assert then the number of bytes specified in the wLength field of the setup packet have been transferred.
1	DR	Indicates a device request on the current endpoint. Will only assert for control endpoints 0 No request pending. 1 Request pending.
0	EOF	Monitors the data flow between the FIFO and the USB device and indicates when the end of a USB packet is written into the FIFO or the USB device as the end of a frame. 0 End of frame (USB packet) was not sent/received. 1 End of frame (USB packet) sent/received.

### 20.3.1.12 Endpoint $n$ Interrupt Mask Register (USB\_EP $n$ IMR)

The endpoint interrupt mask register allows software to mask individual interrupts for each endpoint. Writing a 1 to a bit in this register masks the corresponding interrupt in the USB\_EP $n$ SR register. Writing a 0 unmask the interrupt.

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
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

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	FF	FM	FE	FH	FL	MDR	EOT	DR	EOF
W																
Reset	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1
Address	IPSBAR + 0x1C_0038+( $n$ *0x30)															

**Figure 20-13. USB Endpoint  $n$  Interrupt Mask Register (USB\_EP $n$ IMR)**

### 20.3.1.13 Endpoint $n$ FIFO Data Register (USB\_EP $n$ FDR)

The endpoint FIFO data register is the main interface port for the FIFO. Data which is to be buffered in the FIFO, or has been buffered in the FIFO, is accessed through this register. This register can always access data from the FIFO, independent of the FIFO's transmit or receive configuration. It can be accessed by byte, word, or longword. It is recommended to align all accesses to the most significant byte (big endian) of the data port, using the address of USB\_EP $n$ FDR for byte, word, and longword transactions. However, accessing the data port at USB\_EP $n$ FDR+1,2,3 for bytes or USB\_EP $n$ FDR+2 for words is also acceptable.

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	DATA															
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	DATA															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Address	IPSBAR + 0x1C_003C+( $n$ *0x30)															

**Figure 20-14. USB Endpoint  $n$  FIFO Data Register (USB\_EP $n$ FDR)**

**Table 20-14. USB\_EPnFDR Field Descriptions**

Bits	Name	Description
31–0	DATA	When read, this is the receive FIFO data. When written, this is the transmit FIFO data.

### 20.3.1.14 Endpoint *n* FIFO Status Register (USB\_EPnFSR)

The FIFO status register contains bits which provide information about the status of the FIFO controller. Some of the bits of this register used to generate DMA requests, are provided here for visibility. The bits marked sticky are cleared by writing a one to their position.

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	F0	F1	F2	F3	0	RXW	UF	OF	FR	FULL	ALARM	EMPTY
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

	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
Address	IPSBAR + 0x1C_0040+(n*0x30)															

**Figure 20-15. USB Endpoint *n* FIFO Status Register (USB\_EPnFSR)****Table 20-15. USB\_EPnFSR Field Descriptions**

Bits	Name	Description
31–28	—	Reserved, should be cleared.
27	F0	Read only. Provides frame status indicator. 0 No frame boundary. 1 A frame boundary has occurred on the [31:24] byte of the bus.
26	F1	Read only. Provides frame status indicator. 0 No frame boundary. 1 A frame boundary has occurred on the [23:16] byte of the bus.
25	F2	Read only. Provides frame status indicator. 0 No frame boundary. 1 A frame boundary has occurred on the [15:8] byte of the bus.
24	F3	Read only. Provides frame status indicator. 0 No frame boundary. 1 A frame boundary has occurred on the [7:0] byte of the bus.
23	—	Reserved, should be cleared.

**Table 20-15. USB\_EP $n$ FSR Field Descriptions (Continued)**

Bits	Name	Description
22	RXW	Receive wait condition. Indicates that the FIFO receive bus is incurring wait states because there is not enough room in the FIFO to accept the data without causing overflow. This bit is sticky. 0 No wait state. 1 One or more wait states occur on the FIFO receive bus.
21	UF	Indicates FIFO underflow, sticky bit. 0 No Underflow. 1 Read pointer has passed the write pointer.
20	OF	Indicates FIFO overflow, sticky bit 0 No overflow. 1 Write pointer has passed the read pointer.
19	FR	Frame ready indicator. Only active when the FIFO is programmed for frame mode. All complete frames must be read from the FIFO to clear this alarm. Read only. 0 No complete frames exist in the FIFO. 1 One or more complete frames exists in the FIFO.
18	FULL	FIFO full, read only. 0 The FIFO is not full. 1 The FIFO has requested attention because it is full. The FIFO must be read to clear this alarm.
17	ALARM	FIFO alarm. This read-only bit indicates that the FIFO has determined an alarm condition.  When the FIFO is configured to receive (OUT), the FIFO alarm provides high level indication. The bit will be set when there are less than alarm bytes free in the FIFO. The alarm is cleared when the FIFO is read so that fewer than EP $n$ FCR[GR] bytes remain in the FIFO.  When the FIFO is configured to transmit (IN), the FIFO alarm provides low level indication. The bit will set when there are more than alarm bytes in the FIFO. The alarm is cleared when the FIFO is written so that less than (4 × EP $n$ FCR[GR]) free bytes in the FIFO. 0 Alarm not set. 1 The FIFO has requested attention because it has determined an alarm condition.
16	EMPTY	FIFO empty indicator, ready only 0 The FIFO is not empty. 1 The FIFO has requested attention because it is empty. The FIFO must be written to clear this alarm.
15-0	—	Reserved, should be cleared.

### 20.3.1.15 Endpoint $n$ FIFO Control Register (USB\_EP $n$ FCR)

The FIFO control register provides programmability of FIFO behaviors, including last transfer granularity and frame operation. Last transfer granularity allows the user to control when the FIFO controller stops requesting data transfers through the FIFO alarm by modifying the deassertion point of the alarm, ensuring the data stream is stopped at a valid point, or there remains enough space in the FIFO to unload the input data pipeline.

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	WFR	0	FRAME	GR			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
Address	IPSBAR + 0x1C_0044+(n*0x30)															

**Figure 20-16. USB Endpoint *n* FIFO Status Register (USB\_EP*n*FSR)****Table 20-16. USB\_EP*n*FCR Field Descriptions**

Bits	Name	Description
31–30	—	Reserved, should be cleared
29	WFR	Write Frame. When this bit is set, the FIFO controller assumes the next write to its data port is the end of a frame, and will tag the incoming data accordingly. This bit is automatically cleared by a write to the data port. 0 Next write to FIFO data register is not the end of frame. 1 Next write to FIFO data register is the end of frame.
28	—	Reserved, should be cleared.
27	FRAME	Frame mode enable. When this bit is set, the FIFO controller monitors frame done information from the peripheral. Setting this bit also enables the other frame control bits in this register, as well as other frame functions. This bit must be set to use frame functions. In FRAME mode, the FIFO uses its internal frame pointer and information from the peripheral to transfer only full frames of data, as defined by the peripheral. Since the controller only keeps a pointer to the end of the last complete frame, a read request may contain more than one frame of data. 0 Frame mode disabled. 1 Frame mode enabled.
26–24	GR	These bits define the deassertion point for the “high” service request, and also define the deassertion point for the “low” service request. A “high” service request is deasserted when there are less than GR[2:0] data bytes remaining in the FIFO. A “low” service request is deasserted when there are less than (4 * GR[2:0]) free bytes remaining in the FIFO. The direction, type, and packet size are defined in the USB_EP <i>n</i> SR registers.
23–0	—	Reserved, should be cleared.

### 20.3.1.16 Endpoint *n* Last Read Frame Pointer (USB\_EP*n*LRFP)

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
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
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	LRFP											
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Address	IPSBAR + 0x1C_0048+( <i>n</i> *0x30)															

**Figure 20-17. USB Endpoint *n* Last Read Frame Pointer (USB\_EP*n*LRFP)**

**Table 20-17. USB\_EP*n*LRFP Field Descriptions**

Bits	Name	Description
31–12	—	Reserved, should be cleared.
11–0	LRFP	<p>A FIFO-maintained pointer which indicates the location of the start of the most recently read frame, or the start of the frame currently in transmission. The LRFP updates on FIFO read data accesses to a frame boundary. The LRFP can be read and written for debug purposes. When FRAME is not set, then this pointer has no meaning. The last read frame pointer is reset to zero, and non-functional bits of this pointer are unaffected by reset.</p> <p>The width of the LRFP field is dependent on the size of the FIFO. The low order bit of this field is always at bit position zero. The high order bit is dependent on the size of the FIFO. Bit positions above the FIFO size are treated as undefined and are unaffected by read or write operations. The maximum field width is twelve bits (i.e. FIFO size of <math>2^{12}</math> with a maximum value of 0xFFF).</p>

### 20.3.1.17 Endpoint *n* Last Write Frame Pointer (USB\_EP*n*LWFP)

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
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
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	LWFP											
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Address	IPSBAR + 0x1C_004C+( <i>n</i> *0x30)															

**Figure 20-18. USB Endpoint *n* Last Write Frame Pointer (USB\_EP*n*LWFP)**

**Table 20-18. USB\_EP $n$ LWFP Field Descriptions**

Bits	Name	Description
31–12	—	Reserved, should be cleared.
11–0	LWFP	<p>A FIFO-maintained pointer which indicates the start of the last frame written into the FIFO. The LWFP updates on FIFO write data accesses which create a frame boundary, whether that be by setting the WFR bit in the FIFO Control Register, or by feeding a frame bit in on the appropriate bus. The LWFP can be read and written for debug purposes. Data between the LWFP and write pointer is of an incomplete frame, while data between the read pointer and the LWFP has been received as whole frames. When FRAME is not set, then this pointer has no meaning. The last write frame pointer is reset to zero, and non-functional bits of this pointer are unaffected by reset.</p> <p>The width of the LWFP field is dependent on the size of the FIFO. The low order bit of this field is always at bit position zero. The high order bit is dependent on the size of the FIFO. Bit positions above the FIFO size are treated as undefined and are unaffected by read or write operations. The maximum field width is twelve bits (i.e. FIFO size of <math>2^{12}</math> with a maximum value of 0xFFF).</p>

**20.3.1.18 Endpoint  $n$  FIFO Alarm Register (USB\_EP $n$ FAR)**

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
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
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	ALARM											
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Address	IPSBAR + 0x1C_0050 + ( $n \times 0x30$ )															

**Figure 20-19. USB Endpoint  $n$  FIFO Alarm Register (USB\_EP $n$ FAR)**



**Table 20-19. USB\_EPnFAR Field Descriptions**

Bits	Name	Description
31–12	—	Reserved, should be cleared.
11–0	ALRM	<p>Provides high/low level alarm information to the user integration logic. A low level alarm reports lack of data while a high level alarm reports lack of free space. This programmable alarm warns the system when the FIFO is almost full of data (FIFO High), or when the FIFO is almost out of data (FIFO Low). This register is programmed to the upper limit for the number of bytes of data or space in the FIFO before an internal alarm is set. If the amount of data or space in the FIFO is above the indicated amount, the alarm will be set in non-frame mode. In frame mode, the alarm will be set if the amount of data or space in the FIFO is above the amount indicated by the ALRM setting OR if there are end of frame bytes in the FIFO (see frame mode operation). The alarm is cleared when there is less data or space than is programmed in the FIFO granularity field of the FIFO Control register. The number of bits in the alarm pointer register will vary with the address space of the FIFO memory and the alarm pointer is initialized to 0.</p> <p>The width of the ALRM field is dependent on the size of the FIFO. The low order bit of this field is always at bit position zero. The high order bit is dependent on the size of the FIFO. Bit positions above the FIFO size are treated as undefined and are unaffected by read or write operations. The maximum field width is twelve bits (i.e. FIFO size of <math>2^{12}</math> with a maximum value of 0xfff).</p>

**20.3.1.19 Endpoint *n* FIFO Read Pointer (USB\_EPnFRP)**

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
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
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	RP											
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Address	IPSBAR + 0x1C_0054+( <i>n</i> *0x30)															

**Figure 20-20. USB Endpoint *n* FIFO Read Pointer (USB\_EPnFRP)**

**Table 20-20. USB\_EP $n$ FRP Field Description**

Bits	Name	Description
31–12	—	Reserved
11–0	RP	A FIFO-maintained pointer which points to the next FIFO location to be read. The physical address of this FIFO location is actually the sum of the read pointer and the FIFO base, which is provided through a port to the FIFO controller. This base address can vary, but if chosen properly, the FIFO RAM address can be concatenated with the read pointer instead of requiring hardware for addition. The read pointer can be both read and written. This ability facilitates the debug of the FIFO controller and peripheral drivers. The width of the RP field is dependent on the size of the FIFO. The low order bit of this field is always at bit position zero. The high order bit is dependent on the size of the FIFO. Bit positions above the FIFO size are treated as undefined and are unaffected by read or write operations. The maximum field width is twelve bits (i.e. FIFO size of $2^{12}$ with a maximum value of 0xFF).

### 20.3.1.20 Endpoint $n$ FIFO Write Pointer (USB\_EP $n$ FWP)

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
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
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	WP											
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Address	IPSBAR + 0x1C_0058 + ( $n \times 0x30$ )															

**Figure 20-21. USB Endpoint  $n$  FIFO Write Pointer (USB\_EP $n$ FWP)****Table 20-21. USB\_EP $n$ FWP Field Description**

Bits	Name	Description
31–12	—	Reserved
11–0	WP	A FIFO-maintained pointer which points to the next FIFO location to be written. The physical address of this FIFO location is actually the sum of the read pointer and the FIFO base, which is provided through a port to the FIFO controller. This base address can vary, but if chosen properly, the FIFO RAM address can be concatenated with the read pointer instead of requiring hardware for addition. The read pointer can be both read and written. This ability facilitates the debug of the FIFO controller and peripheral drivers.

## 20.4 Functional Description

This section provides a complete functional description of the USB device module, detailing the operation of the design from the end user perspective. All operating modes, and each major block within the design are discussed.

### 20.4.1 USB Components

Each component shown in the block diagram from [Section 20.1.1, “Block Diagram”](#) is discussed within this section.

#### 20.4.1.1 Descriptor RAM

This 1-Kbyte descriptor RAM allows the programmer to preload the descriptors and modify them as necessary. Refer to chapter 9 of the USB specification for more information on the descriptor structures.

#### 20.4.1.2 FIFO Controller

The USB never needs to communicate with more than one data FIFO at a time. Therefore, the FIFO controller modules in the USB device share a single RAM for FIFO data storage. User programmable registers allow on-the-fly configuration of individual FIFO sizes.

The USB protocol has some specialized requirements which affect FIFO implementation and create a need for one FIFO per USB endpoint. The USB host may access any endpoint on the device in any order, even the same endpoint in back to back transactions. There is also a latency requirement. The USB device must respond to the USB host within a certain number of USB bit times or the device loses its time slice on the USB until some point in the future. In order to achieve maximum USB bandwidth utilization, the USB device must be able to provide full packets of data to the USB host immediately upon request and receive full packets from the host on request. This requirement results in one FIFO per USB endpoint. Depending upon the traffic requirements, the user may need to size the FIFO to support double buffering.

## 20.5 Reset Strategy

This section describes how to reset the device. Descriptions are limited to sources and requirements of reset. All sources of reset are shown in [Table 20-22](#).

**Table 20-22. Reset Summary**

Reset Type	Source	Characteristics
Hard Reset	Internal Peripheral Bus	Resets all storage elements in the USB device.
USB Device Reset	USB_CR Register	"Soft" reset of USB device. Control registers and FIFO's are left intact. Required any time a connect or disconnect event occurs on the USB that is not accompanied by a hard reset.
USB Bus Reset	USB Cable	"Soft" reset of all devices on USB so that the host may re-enumerate them with new addresses. Does not reset any datapath elements, but may require user to flush FIFO's and prepare to configure a change request from USB host.

## 20.5.1 Description of Reset Operation

This USB device includes three reset modes: hard reset, USB device reset, and USB bus reset. All three reset modes are described in this section.

## 20.5.2 Hard Reset

A hard reset is generated from the USB device's bus interface and resets all storage elements in both the external PHY and in the USB device. A hard reset also issues a USB device reset.

### 20.5.2.1 USB Device Reset

The USB device must be reset any time a connect/disconnect occurs on the USB system. Any time the device is plugged in or unplugged from the USB system, software must initiate a USB device reset in order to ensure that the device can properly communicate with the USB host. Reset signalling is discussed in Chapter 7 of the USB specification. A USB device reset may invalidate data remaining in the data FIFOs. Depending upon the application, software may need to flush the data FIFOs before proceeding.

### 20.5.2.2 USB Bus Reset

USB bus reset signalling can occur on the USB for a number of reasons. When the device receives bus reset signalling, it means that the host is preparing to re-enumerate the bus. All transactions in progress must be invalidated, and the device software should prepare to receive configuration changes.

Because the device may have valid, but as yet unread data remaining in the FIFOs when USB bus reset occurs, the hardware does not flush the FIFOs. When device software receives reset signalling, it should first finish reading any unread data from the FIFOs, then execute FIFO flush operations on all of the FIFOs. This guarantees that the datapath is empty and ready for new data transfer operations when reset signalling and re-enumeration are complete.

Reset signalling is discussed in Chapter 7 of the USB specification.

## 20.6 Interrupts

This section describes interrupts originated by the USB device. The available interrupts are briefly described in [Table 20-23](#).

**Table 20-23. Interrupt Summary**

Number of Interrupt	Offset	Vector	Priority	Source	Description
INT43–INT46	N/A	N/A	N/A	USB Endpoints USB_EPnISR	USB Endpoint 0-3 specific interrupts. Tied to FIFO status and datapath logic.
INT47	N/A	N/A	N/A	USB Global Interrupt USB_ISR	USB device global status interrupt.

### 20.6.1 Description of Interrupt Operation

This USB device supports interrupts from two sources: the USB endpoints and the USB device's control logic. Both interrupt sources behave exactly the same way.

Each interrupt source is associated with an interrupt status register and an interrupt mask register.

At reset time, all interrupts are masked. Clearing a bit in the mask register unmask the corresponding interrupt, while setting the bit masks the interrupt. When an interrupt event occurs and the interrupt is unmasked, an interrupt request asserts to the CPU's interrupt controller. Re-masking the interrupt without actually clearing the request will turn-off the interrupt request to the CPU, but will not affect the status of the pending interrupt.

The CPU services an interrupt by reading the appropriate interrupt status register and clearing the pending interrupts. Interrupts are cleared by writing a 1 to the appropriate bit in the interrupt status register. Writing zeros to the interrupt status registers has no effect. Once all pending interrupts have been cleared, the interrupt request to the CPU will also clear.

### 20.6.2 Detailed Interrupt Descriptions

A number of interrupts are generated by the USB device to indicate situations requiring attention from the CPU. The types of interrupts are broken into two categories: the USB global interrupt and endpoint specific interrupts.

#### 20.6.2.1 USB Global Interrupt

The USB global interrupt indicates such things as global configuration and status changes that pertain to the USB, which are described below. Refer to [Section 20.3.1.7, “USB Interrupt Status Register \(USB\\_ISR\)”](#) for more information.

### **20.6.2.1.1 Missed Start of Frame (MSOF)**

The MSOF interrupt is used to inform software that it did not service the SOF interrupt before another SOF was received.

### **20.6.2.1.2 Start of Frame (SOF)**

The SOF interrupt indicates that a start of frame token was received by the device. The current USB frame number may be read from the USB\_FRAME register. The start of frame interrupt is usually used by isochronous devices to provide a stable timebase.

### **20.6.2.1.3 End of USB Reset Signalling (RESET\_STOP)**

The RESET\_STOP interrupt indicates that reset signalling from the host on the USB has stopped. When not asserted, RESET\_STOP indicates that no end of reset event has been received.

### **20.6.2.1.4 Start of USB Reset Signalling (RESET\_START)**

The RESET\_START interrupt indicates that reset signalling on the USB has begun. When reset signalling is active on the USB, the device should not expect to receive any transactions on the bus. This interrupt results in a reset of the USB device into the powered state. See the USB specification for more information on the powered state. The powered state means that the USB device will not respond to USB transactions.

This interrupt only indicates the start of reset signalling. Status of the USB at any given moment can be verified by examining the USB\_SR register.

The user is cautioned that presence of USB reset signalling invalidates any transaction in progress. On detection of RESET\_START, user software should read any remaining valid data from receive FIFOs and flush all the others. When reset signalling is detected, user software must clear any pending interrupts and ensure that the USB device is configured properly after the reset.

### **20.6.2.1.5 Resume Signalling Detected (RES)**

This interrupt asserts when resume signalling is detected on the USB. The device uses this interrupt to wake up from suspend mode and resume normal operations.

### **20.6.2.1.6 USB Suspended (SUSP)**

This interrupt asserts when the USB goes into suspend mode. Suspend mode occurs when the device fails to receive any traffic from the USB for a period of 3ms. Once suspend mode is detected, the device should put itself into a low power standby mode.

### **20.6.2.1.7 Match Detected in USB\_FRAME (FM)**

This interrupt asserts when the frame number programmed into the MATCH field of the USB\_FRAME register is the same as the FRAME field of the USB\_FRAME register. Isochronous pipes may use this interrupt for synchronization between the data source and sink.

### **20.6.2.1.8 USB Configuration Change (CC)**

This interrupt indicates that the USB host selected a different configuration or alternate interface from the list of descriptors stored in the descriptor RAM module. Software should read the USB\_SR register to determine which configuration and alternate interface are current, then reconfigure itself accordingly.

## **20.6.2.2 Endpoint Interrupts**

The endpoint interrupts indicate requests for service by specific USB endpoints. All bits are maskable. When an event occurs that causes an interrupt condition to occur, and the corresponding bit in the interrupt mask register is 0, an interrupt signal will assert on the USB device's interface. Writing a 1 to the associated bit in the interrupt status register clears the interrupt.

### **20.6.2.2.1 FIFO Full & Empty Indicators (FF & FM)**

These interrupts assert when the FIFO is either full or empty.

### **20.6.2.2.2 FIFO Error Indicator (FE)**

This interrupt indicates some abnormal condition occurred in the FIFO. The cause of the error can be verified by reading the USB\_EPFSRFSR register associated with the FIFO that had the error.

### **20.6.2.2.3 FIFO High & Low Level Alarms (FH & FL)**

Each FIFO has an alarm register, USB\_EPnFAR. If the byte count in the FIFO is above the level specified by the alarm register, then the FH interrupt will assert. If the byte count in the FIFO is below the level specified by the alarm register, then the FL interrupt will assert.

### **20.6.2.2.4 End of Transfer (EOT)**

This interrupt asserts after the last data byte of a USB transfer has been received or sent by a USB device. The end of a USB transfer is indicated by either a zero byte packet, or by a data packet shorter than the maximum packet size for the endpoint. Listed below are other instances in which the EOT interrupt will or will not assert:

- The EOT interrupt will generally assert along with an EOF interrupt, although it is possible to get an EOT interrupt without an EOF interrupt if a transfer is terminated on a USB packet boundary.
- The EOT interrupt never asserts for setup packets.
- The EOT interrupt asserts after every interrupt packet transfer, at the completion of bulk data transfers, and at the end of the data phase of a control transfer.
- The EOT interrupt only asserts for isochronous packet transfers if the USB device reports that the packet data is error free. This can be used along with the EOF interrupt to determine when a transfer error of some sort occurs on an isochronous endpoint.

#### **20.6.2.2.5 Device Request (DR)**

The device request (setup packet) interrupt indicates that the most recently received packet was a setup or device request packet. Software on the USB device must decode and respond to the packet in order to complete a vendor, class, or standard request.

#### **20.6.2.2.6 Multiple Device Request (MDR)**

The multiple device request interrupt asserts when two or more setup packets have been received before the DR interrupt was cleared. This interrupt is used to determine when the USB host has aborted a transfer in progress. In this case, the device will receive a setup packet, followed by a new setup packet before it has completed processing of the original command.

#### **20.6.2.2.7 End of Frame (EOF)**

This interrupt indicates that an end of frame marker was sent or received on the FIFO/USB interface. This interrupt will assert if a DR is received. This interrupt asserts for bulk, control, isochronous, and interrupt data. While packet retries are not supported for isochronous endpoints, the end of frame indicator is still valid and may be used along with the SOF interrupt to control data flow.

### **20.6.2.3 Interrupts, Missed Interrupts, and the USB Device**

There are a number of cases where improper operation of the device could result if interrupts are not serviced in a timely manner. For example, if a CC interrupt is received, the device does not service it, and another CC interrupt is received. This could leave the device in an incorrect operating mode. The interrupts of concern are SOF, CC, EOT, and DR.

The missed-interrupt behaviors are listed below:

#### **20.6.2.3.1 Start of Frame (SOF)**

If the device misses a start of frame interrupt, the MSOF bit asserts in the USB\_ISR register.



### 20.6.2.3.2 Configuration Change (CC)

When the USB device receives a CC interrupt, the device will acknowledge (NAK) all traffic from the USB host on all endpoints until software clears the interrupt bit. This prevents the device configuration from getting out of sync with what the host has requested.

### 20.6.2.3.3 End of Transfer (EOT)

When an end of transfer is received on a BULK OUT endpoint, the device will not NAK all traffic on that endpoint until software clears the interrupt bit. This prevents data from two different transfers from becoming mixed up in a FIFO.

### 20.6.2.3.4 Device Request (DR)

When a device request is received, the device will NAK all in/out traffic on the affected endpoint until software clears the interrupt bit. This ensures that the device correctly identifies the setup packet in the FIFO and can clear the FIFO before the data phase is allowed to begin. If multiple setup packets are received, the MDR interrupt will assert to warn of that condition. The control transfer state machine within the USB device will not advance from the setup phase to the data phase until the DEVREQ bit is cleared.

## 20.7 Software Interface

This section provides information pertinent to programming the device. It includes information to configure, program, and debug the device.

### 20.7.1 Device Initialization

During device initialization, user software downloads critical configuration information to the USB device's descriptor RAM datapath for processing. This process is performed at two different times: hard reset and when the device is first connected to the USB system. The USB connection event is described in the *USB Specification, Revision 2.0, Chapter 7, "Electrical Specification."*

At power-up time, the USB device contains no configuration information. The USB device does not know how many endpoints it has available or how to find the descriptors. Initialization for the device consists of downloading this information to the appropriate memories and configuring the datapath to match the intended application. The following steps are involved in the initialization process:

1. Perform a hard reset or a USB device reset (USB\_CR[RST]). Software must wait for the USB\_DAR[CFG] bit to assert before attempting to communicate with the USB device.
2. Download configuration data (StringBufs, CfgBufs, and EpBufs) to the device (USB\_DDR).

3. Download USB descriptors to the descriptor RAM via the USB\_DAR and USB\_DDR registers.
4. Program the USB Interrupt Mask register (USB\_IMR) to enable interrupts not associated with a particular endpoint.
5. Program each endpoint's control and interrupt registers (USB\_EPnIMR and USB\_EPn\_CTRL) to support the intended data transfer modes.
6. Select the FIFO sizes (USB\_MCR).
7. Program endpoint type, direction, and maximum packet size in the USB\_EPnSR register for each endpoint.
8. Program the FIFO controller registers. For each enabled endpoint, program frame mode, granularity, alarm level, frame size, etc. (USB\_EPnFCR). Normally, all endpoints should be programmed with FRAME mode enabled.
9. Enable the USB for processing (USBENA bit in USB\_CR).

Note that USB device initialization is a time critical process. The USB host will wait about 100ms after power-on or a connection event to begin enumerating devices on the bus. This device must have all of the configuration information available when the host requests it.

Once the device has been enumerated, the USB host will select a specific configuration and set of interfaces on the device. Software on the device must be aware of USB configuration changes in order to maintain proper communication with the USB host. The software retains sole responsibility for knowing which configuration and alternate interface are current at any given time. The CFG\_CHG interrupt in the USB\_ISR register reports changes in device configuration and alternate interface settings to the software. Note that the software is required to respond to the CFG\_CHG interrupt. To prevent the state of the device from becoming out of sync with respect to the host, the device halts further traffic on the USB while this interrupt is pending.

### 20.7.1.1 Configuration Download

The configuration download process initializes a number of buffers within the USB device that define its personality on the USB system. This information includes RAM addresses for the configuration (CfgBufs), string descriptors (StringBufs), and one 40 bit buffer per endpoint (EpBufs).

**Table 20-24. Configuration Buffers Format (CfgBufs)**

Bits	Type	Description
31–16	Descriptor Size	This is the size of the Descriptor to be returned when the USB host issues a Get_Descriptor command to the device. For the MCF5275 the minimum descriptor size that can be set is 46 bytes (with one endpoint) and the maximum can be 60 bytes. This includes the combined length of all descriptors (configuration, interface, endpoint and class or vendor specific) returned for this configuration.
15–0	Descriptor Address Pointer	This is the address pointer of where the descriptor is located in the USB descriptor RAM. The address pointer must be set within 0x0000 and 0x0300.

**Table 20-25. String Buffers Format (StringBufs)**

Bits	Type	Description
23–16	String Length	This is the length of the string in bytes with which this String Buffer is associated. This is the USB blength of the String Descriptor for the String.
15–0	String Address Pointer	This is the address pointer of a location in the USB descriptor RAM at which the String is located.

The endpoint buffers are 40 bit-per-endpoint data strings that are loaded directly into the USB device. They associate “logical” endpoint numbers in the USB software stack with hardware within the USB device. Specifically, they attach each endpoint to a USB configuration, interface, and alternate interface. Then they specify transfer type, packet size, and data direction among other characteristics. In addition, each endpoint buffer includes a 16 bit data field that allows the end user to specify the mapping between USB endpoints and the hardware FIFOs for the USB device.

**Table 20-26. Endpoint Buffer Format (EpBufs)**

Bits	Type	Description
39–36	EpNum	Logical Endpoint Number
35–34	Ep_Config	Configuration number to which the endpoint belongs.
33–32	Ep_Interface	Interface number to which the endpoint belongs.
31–30	Ep_AltSetting	Alternate setting to which the endpoint belongs.
29–28	Ep_Type	Type of endpoint. 00 Control 01 Isochronous 10 Bulk 11 Interrupt
27	Ep_Dir	Direction of endpoint. Ignored for control endpoints. 0 Out endpoint 1 In endpoint
26–17	Ep_MaxPktSize	Maximum packet size for this endpoint.

**Table 20-26. Endpoint Buffer Format (EpBufs)**

Bits	Type	Description
16	Unused	Unused bit. Set this bit to zero.
15–0	Ep_BufAdrPtr	16 bit address that points to one of the four FIFOs of the USB device. See <a href="#">Section 20.7.1.2, “USB Endpoint to FIFO Mapping,”</a> for more information on the format for this field.

The USB device has one configuration, one interface with two alternate settings, three strings, and four logical endpoints that are configured as seven physical endpoints.

Download of the configuration data consists of the following steps:

1. Verify that the CFG bit is set in the USB\_DDR register. This ensures that the USB device has been properly reset and is prepared to take data.
2. Write the configuration buffers, string buffers, and endpoint buffers to the USB\_DDR register. These buffers should be written most significant byte first. After writing each byte, and before performing any other operation on the peripheral, check that the BSY bit has cleared in the USB\_DAR register.
3. After writing all endpoint buffer configuration bytes, check the CFG bit of USB\_DAR to verify that the configuration load has completed. This bit changes from 1 to 0 after the last byte has been loaded into the USB device.

#### 20.7.1.1.1 Logical vs. Physical Endpoints

There is a distinction in the USB device between logical and physical endpoints. Logical endpoints are those seen by the host. Physical endpoints can have the same logical endpoint number but different characteristics, such as being different direction or existing on a different alternate interface. Endpoint 0 is always a control endpoint. Endpoints 1, 2, and 3 can be organized on one or both alternate interfaces. See for information about configuring logical endpoints.

#### 20.7.1.2 USB Endpoint to FIFO Mapping

The MCF5275 USB device recognizes up to four logical endpoints or seven physical endpoints. Some hardware must be associated with each physical endpoint. Since two physical endpoints mapped on a logical endpoint cannot be used simultaneously, one hardware FIFO will be sufficient to support both.

Each logical endpoint consists of a FIFO that can be programmed independently for depth, direction, transfer type, frame mode, and low/high alarms. In order to fit these hardware endpoints to the USB’s “logical” endpoints, a mapping is established in the USB device’s endpoint buffer.

Endpoint type, direction, and packet size are defined via the USB\_EPnSR register for each endpoint. FIFO characteristics are programmed via the USB\_EPnFCR and USB\_EPn\_ALRM registers. Mapping of USB endpoints to specific hardware FIFOs is accomplished at power-up

time when the USB device endpoint buffers are downloaded to the device from the CPU. Each endpoint buffer makes 16 bits available to the user for the purpose of identifying a hardware endpoint.

**Table 20-27. Ep\_BufAdrPtr Format**

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	TRXTYP[1:0]		RESERVED										EPNUM[3:0]			

**Table 20-28. Ep\_BufAdrPtr Field Descriptions**

Bits	Name	Description
15–14	TRXTYP	For this revision of the USB device, these bits must be set to 0b00 for endpoint 0, and 0b11 for all other endpoints.
3–0	EPNUM	This field maps the endpoint to one of the USB device's hardware FIFOs. Multiple USB device endpoints may map to a single hardware FIFO. It is up to the software to monitor and control any data hazards related to operation in this way.

Ep\_Buf0 is a special case. The value in Ep\_BufAdrPtr indicates the descriptor RAM address instead of one of the hardware endpoints (FIFOs). Logical and physical endpoint 0 always use hardware endpoint 0 for transfers not handled by the USB device. Software can map the remaining logical endpoints to any of the remaining three hardware endpoints.

### 20.7.1.3 USB Descriptor Download

The USB descriptors are standard data structures which are used by the USB host to allocate bandwidth and to figure out how many and what kind of endpoints are available on the device. While this data is stored in the device's memory space, it is not directly used by either the USBCORE or the USB devices. The data gets returned to the USB host during a GET\_DESCRIPTOR request. The USB host picks a specific configuration and alternate interface, and then instructs the device which to enable. The USB descriptor formats are defined in Chapter 9 of the USB specification. Software programs are available from various sources to assist in creating the descriptor tables.

Any time the USB host changes the USB device configuration or alternate interface, an interrupt is generated in this device to inform the user that something changed. Software must then read the USB status register (USB\_SR) to determine the new current configuration and alternate setting. Software is responsible for FIFO management and endpoint re-configuration each time the USB configuration changes.

Download of the descriptor data consists of the following steps:

1. Verify that the CFG bit is clear in the USB\_DDR register. This ensures that the USB device has been properly reset and configured and is prepared to take data.

2. Write the starting address of the descriptors into the DADR field of the USB\_DAR register. The address written to this register is the address of the descriptors within the descriptor RAM attached to the USB device, not the system RAM.
3. Write each byte of the descriptor table to the DDAT field of the USB\_DDR register. The USB\_DAR register increments automatically at each register access (read or write).

#### 20.7.1.4 USB Interrupt Register

If the application makes use of the interrupt registers, then the specific interrupts to be used must be enabled. During a reset, all interrupts revert to the masked state. The USB global interrupt (affecting the whole USB device) is programmed separately from those affecting a single endpoint.

#### 20.7.1.5 Endpoint Registers

For each endpoint, the characteristics of the FIFO and a number of interrupt sources may be programmed. Software must program the following registers:

- USB endpoint interrupt mask (USB\_EPnIMR)
- Separate interrupt registers are provided for each hardware FIFO. Enable the interrupts pertaining to the application by writing a 0 to the mask bit for that interrupt.
- Endpoint FIFO controller configuration(USB\_EPnFCR).
- Each FIFO is programmed based for the type of data transmission used by the endpoint.
- FIFO alarm register (USB\_EPn\_ALRM).
- For bulk traffic (FRAME=1), the alarm level is normally programmed to a multiple of the USB packet size(i.e. for 8 byte packets and a 16 byte FIFO, the alarm would be programmed to 8 bytes) to allow the request lines to request full packets. For single buffered endpoints (packet size =8, FIFO depth=8 bytes), the alarm is normally programmed to 0. For isochronous traffic, the alarm is programmed to allow streaming operation to occur on the isochronous endpoint.

#### 20.7.1.6 FIFO Sizes

FIFO sizes must be programmed to match the traffic sent across the USB. The USB\_MCR register allows software to choose which memory configuration is to be used at any given time.

#### 20.7.1.7 Enable the Device

The last step in initializing the USB device is to enable it for processing via the USB\_CR register. Set the USBSPD bit appropriately for full or low speed operation(full speed operation) and set the USBENA (module enable) bit.

Note that the USBSPD bit must not be changed after the initial USB device configuration, otherwise unpredictable operation of the USB may result.

## 20.7.2 Exception Handling

Exception handling refers to two situations. The first occurs when corrupted frames must be discarded. The second is when an error situation occurs on the USB.

Corrupted frames are automatically discarded by the hardware. No software intervention is required to deal with this problem.

If the device cannot respond to the host in the time allotted, hardware automatically handles retries. No software intervention is required in this case.

The following types of error situations can arise and must be dealt with by the software:

### 20.7.2.1 Unable to Complete Device Request

In the event that the software receives a device request that it cannot interpret, it asserts the CMD\_ERROR and CMD\_OVER bits back to the USB device. This results in a STALL to the endpoint in question and requires intervention from the USB host to clear. When the CMD\_OVER bit clears, it indicates that the USB host has cleared the stall condition.

### 20.7.2.2 Aborted Device Request

When the host sends a setup packet to the device, it is possible that the ACK handshake from the device could be corrupted, and lost on its way back to the host. In this case, the host will retry the setup packet. In this case, the device can wind up with two or more setup packets in its FIFO. There are two ways to detect this condition:

- the presence of MDEVREQ interrupt
- the SIP bit in the endpoint status register (USB\_EPnSR) is active.

In either case, the presence of more than one setup packet causes the device to invalidate all packets except the last one in the FIFO. Once a packet is acknowledged, no more are sent.

In the case that MDEVREQ is active, software must discard the first packet, and process the second one. In the case of SIP active, software must discard the first one, clear the device request, and wait for the device request to reassert.

### 20.7.2.3 Unable to Fill or Empty FIFO Due to Temporary Problem

If the USB device is unable to fill or empty a FIFO due to a temporary problem (for example the operating system did not service the FIFO in time and it overflowed), the software should stall the endpoint via the STALL bit in the endpoint control register. This will abort the transfer in progress and force intervention from the USB host to clear the stall condition. The STALL register bit

automatically clears once the stall condition takes effect. It is up to the application software on the host to deal with the stall condition and notify the device as to how it should proceed.

#### **20.7.2.4 Catastrophic Error**

In the case of a catastrophic error, the software should execute a hard reset, reinitialize the USB device, and wait for the USB host to re-enumerate the bus.

### **20.7.3 Data Transfer Operations**

Three types of data transfer modes exist for this USB device: control transfers, bulk transfers, and isochronous transfers. In addition to the three data transfer modes listed, the USB specification also supports an interrupt transfer. From the point of view of the USB device, the interrupt transfer type is identical to the bulk data transfer mode, and no additional hardware is supplied to support it. This section covers the transfer modes and how they work from the ground up.

All data is moved across the USB in terms of packets. Groups of packets are combined to form data transfers. The same packet transfer mechanism applies to bulk, interrupt, and control transfers. Isochronous data is also moved in the form of packets, but since isochronous pipes are given a fixed portion of the USB bandwidth at all time, there is no concept of an end of transfer.

#### **20.7.3.1 USB Packets**

Data moves across the USB in units called packets. Packets range in size from 0 to 1023 bytes, and, depending on the transfer mode the packet size, is restricted to a small set of values. For bulk, control, and interrupt traffic, packet sizes are limited to 8, 16, 32, or 64 bytes. Isochronous data packets can take any size from 0 to 1023 bytes. The packet size is programmable within the USB device on an endpoint by endpoint basis.

The terms packet and frame are used interchangeably within this document. While USB traffic occurs in units called packets, the FIFO mechanism uses the term frames for the same blocks of data. The only difference between frames and packets from the user's standpoint is that packets may be as little as zero bytes in length, while a frame must be at least one byte in length.

##### **20.7.3.1.1 Short Packets**

Each endpoint has a maximum packet size associated with it. In most cases, packets transferred across an endpoint will be sent at the maximum size. Since the USB does not indicate a data transfer size, or include an end of transfer token, short packets are used to mark the end of data. Software indicates end of data by writing short packets into the FIFO. Incoming short packets are indicated by examining the length of a received packet or by looking at the end of transfer and end of frame interrupts.



### 20.7.3.1.2 Sending Packets

To send a packet of data to the USB host using programmed I/O, use these steps:

1. For an N byte packet, write the first N-1 bytes to the FIFO data register (USB\_EPnFDR). Data may be written as bytes, words, or longwords.
2. For the N<sup>th</sup> byte, set the WFR bit in the USB\_EPnFCR register, then write the last data byte to the USB\_EPnFDR register. Note that data is written in big-endian format. The most significant byte(byte 0) is held in bits 31 to 24.

### 20.7.3.1.3 Receiving Packets

To receive a packet of data from the USB host, either DMA or programmed I/O may be used.

For programmed I/O, follow these steps:

1. Monitor EOF interrupt for the endpoint.
2. On receiving EOF interrupt, prepare to read a complete packet of data. Clear the EOF interrupt so that software will receive notification of the next frame.
3. Read the USB\_EPnFDR register to read in the next piece of data.
4. Read the USB\_EPnFSR register to get the end of frame status bits. If the end of frame bit is set for the current transfer, then stop reading data.
5. Go back to step 3.

### 20.7.3.1.4 Programming the FIFO Controller

The FIFO controller module has two modes of operation, frame and non-frame. For the USB application, normally only frame mode is used.

In frame mode, the FIFO controller can handle automatic hardware retry of bad packets. This mode is used for bulk, control, and interrupt endpoints. During device initialization, the user should configure the FIFOs via the USB\_EPnFCR register for FRAME mode. Data flow is controlled with the end of frame (EOF) and end of transfer (EOT) interrupts, or with the DMA request lines.

## 20.7.3.2 USB Transfers

Data transfers on the USB system are composed of one or more packets. Instead of maintaining a transfer count, the USB host and device send groups of packets to each other in units called transfers. In a transfer, all packets are the same size, except the last one. The last packet in a transfer will be a short packet, as small as 0 bytes in the case that the last data byte ends on a packet boundary.

This section describes how data transfers work from both the device to the host, and from the host to the device.

### 20.7.3.2.1 Data Transfers to the Host

Given an arbitrary sized block of data to be sent to the host, break it into a number of packets sized at the maximum packet size of the target endpoint.

If number of bytes in the transfer is evenly divisible by the number of bytes in the maximum packet size, then the transfer ends on a packet boundary. A zero length packet will be required to terminate the transfer. If the number of bytes in the transfer is not evenly divisible by the maximum packet size, then the last packet of the transfer will be a short packet and no zero length packet is required.

For each packet in the transfer, write the data to the USB\_EPnFDR register. The last byte in each packet must be tagged with the end of frame marker via the USB\_EPnFCR register or comm service request lines. Monitor the FIFO\_LOW interrupt, EOF interrupt, or comm service requests to determine when the FIFO can accept another packet.

After the last byte of the transfer has been written to the FIFO, if a zero length packet is required to terminate the transfer, then set the ZLPS bit for the endpoint.

Wait for the EOT interrupt to determine when the transfer has completed.

### 20.7.3.2.2 Data Transfers to the Device

The length of data transfer from the host is generally not known in advance. The device receives a continuous stream of packets and uses the EOT interrupt to determine when the transfer ended.

Software on the device monitors the EOF interrupt and/or the comm DMA requests to manage packet traffic. Each time a packet is received, the device must pull the data from the FIFO. Each time an end of frame is transferred from the USB device into the data FIFO, the EOF interrupt asserts. At the end of a complete transfer, the EOT interrupt asserts. Until the CPU has serviced the EOT interrupt, the device will NAK any further requests from the host. This guarantees that data from two different transfers will never get intermixed within the FIFO.

### 20.7.3.3 Control Transfers

The USB host sends commands to the device via control transfers. Control transfers may be addressed to any control endpoint. Control transfers consist of up to three distinct phases. Each control transfer begins with a setup phase, followed by an optional data phase, and is completed with a status phase. Command processing occurs in the following steps:

1. SETUP packet received on control endpoint. DEVREQ, and EOF interrupts assert for that endpoint.
2. Read 8 bytes of the setup packet from the appropriate FIFO data register and decode the command.
3. Clear EOF and DEVREQ interrupts.

4. If a data transfer is implied by the command, set up and perform the data transfer. Be careful not to send back more bytes to the USB host than were requested in the wLength field of the SETUP packet. Hardware does not check for incorrect data phase length. The EOT interrupt will assert on completion of the data phase.
5. Assert CMD\_OVER/CMD\_ERROR bits to indicate processing or error status. The USB device will generate appropriate handshakes on the USB to implement the status phase. CMD\_OVER automatically clears at the end of the status phase.
6. Wait for CMD\_OVER to clear, indicating that the device request has completed.

The MCF5275 USB device will handle all of the standard requests without software intervention. User software does not need to be concerned with handling any of the so-called "Chapter 9" requests listed in the USB specification, except for SYNCH\_FRAME.

#### **20.7.3.4 Bulk Traffic**

Bulk traffic guarantees the error-free delivery of data in the order that it was sent, but the rate of transfer is not guaranteed. Bandwidth is allocated to bulk, interrupt, and control packets based on the bandwidth usage policy of the USB host.

##### **20.7.3.4.1 Bulk OUT**

For OUT transfers (from host to device), internal logic marks the start of packet location in the FIFO. If a transfer does not complete without errors, the logic will force the FIFO to back up to the start of the current packet and try again. No software intervention is required to handle packet retries.

User software reads packets from the FIFOs as they appear and stops when an EOT interrupt is received. To enable further data transfers, software services and clears the pending interrupts (EOF or EOT), then waits for the next transfer to begin.

##### **20.7.3.4.2 Bulk IN**

For IN transfers (from device to host), software tags the last byte in a packet to mark the end of frame. If a transfer does not complete without errors, hardware will automatically force the FIFO to back up to the start of the current packet and re-send the data. User software is expected to write data to the FIFO data register in units of the associated endpoint's maximum packet size. The end of frame may be indicated via the WFR bit in the endpoint FIFO control register (USB\_EP $n$ FCR).

In the USB protocol, the last packet in a transfer is allowed to be short (smaller than endpoint's maximum packet size) or even zero length. In order to indicate a zero length packet, the software should set the ZLPS bit in the associated endpoint's control register. The ZLPS bit automatically clears after the zero length packet has been successfully sent to the host.

The EOT interrupt asserts to indicate that the last packet of the IN transfer has completed. Software should clear any pending interrupts (EOT and EOF) to begin the next data transfer.

### 20.7.3.5 Interrupt Traffic

Interrupt endpoints are a special case of bulk traffic. Interrupt endpoints are serviced on a periodic basis by the USB host. Interrupt endpoints are guaranteed to transfer one packet per polling interval. Thus, an endpoint with an 8 byte packet size and serviced every 2 ms would move 32 Kbps across the USB.

The only difference between interrupt transfers and bulk transfers from the device standpoint is that every time an interrupt packet is transferred, regardless of size, the EOT interrupt asserts. The device driver software must service this interrupt before the next interrupt servicing interval to prevent the device from NAK'ing the poll.

Device driver software must be careful that the interrupt endpoint polling interval is longer than the device's interrupt service latency.

### 20.7.3.6 Isochronous Operations

Isochronous operations are a special case of USB traffic. Instead of guaranteeing delivery with unbounded latency, isochronous traffic flows over the bus at a guaranteed rate with no error checking.

#### 20.7.3.6.1 Isochronous Transfers in a Nutshell

The USB host guarantees an endpoint exactly one isochronous packet per frame. Isochronous packets may range from 0 bytes to 1023 bytes. Please refer to the USB specification for more information on isochronous transfer.

Given that isochronous packets may be as large as 1023 bytes, it may not be practical to implement large FIFOs for each endpoint. Instead, the software drivers are responsible for keeping the FIFOs serviced. Each time an IN or OUT request is received on an isochronous endpoint, the software drivers must ensure that the correct amount of data can be transferred without allowing the FIFO to go empty. If the FIFO goes empty during an isochronous packet transfer, the host will terminate the packet immediately and the device loses its time slot until the next USB frame.

In order to allow the driver software to maintain synchronization with the USB host, the USBCORE maintains a register which holds the current USB frame number. An interrupt is asserted each time the frame number changes or when a specific USB frame number is received. The interrupts are maskable.

# Chapter 21

## Watchdog Timer Module

### 21.1 Introduction

The watchdog timer (WDT) is a 16-bit timer used to help software recover from runaway code. The watchdog timer has a free-running down-counter (watchdog counter) that generates a reset on underflow. To prevent a reset, software must periodically restart the countdown by servicing the watchdog.

#### 21.1.1 Low-Power Mode Operation

This subsection describes the operation of the watchdog module in low-power modes and halted mode of operation (by issuing a HALT instruction). Low-power modes are described in [Chapter 8, “Power Management.”](#) [Table 21-1](#) shows the watchdog module operation in the low-power modes, and shows how this module may facilitate exit from each mode.

**Table 21-1. Watchdog Module Operation in Low-power Modes**

Low-power Mode	Watchdog Operation	Mode Exit
Wait	Normal if WCR[WAIT] cleared, stopped otherwise	Upon Watchdog reset
Doze	Normal if WCR[DOZE] cleared, stopped otherwise	Upon Watchdog reset
Stop	Stopped	No

In wait mode with the watchdog control register's WAIT bit (WCR[WAIT]) set, watchdog timer operation stops. In wait mode with the WCR[WAIT] bit cleared, the watchdog timer continues to operate normally. In doze mode with the WCR[DOZE] bit set, the watchdog timer module operation stops. In doze mode with the WCR[DOZE] bit cleared, the watchdog timer continues to operate normally. Watchdog timer operation stops in stop mode. When stop mode is exited, the watchdog timer continues to operate in its pre-stop mode state.

In halted mode (entered by issuing a HALT instruction) with the WCR[HALTED] bit set, watchdog timer module operation stops. When halted mode is exited, watchdog timer operation continues from the state it was in before entering halted mode, but any updates made in halted mode remain. If the WCR[HALTED] bit is cleared, the watchdog timer continues to operate normally after executing a HALT instruction. This is a debug feature available for the user

## 21.1.2 Block Diagram

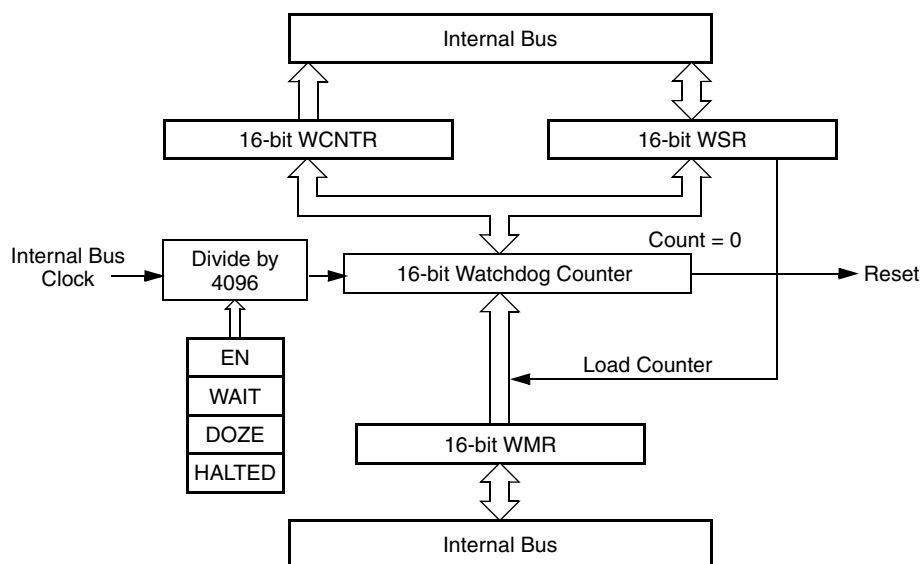


Figure 21-1. Watchdog Timer Block Diagram

## 21.2 Memory Map/Register Definition

This subsection describes the memory map and registers for the watchdog timer. The watchdog timer has a IPSBAR offset for base address of 0x14\_0000. Refer to [Table 21-2](#) for an overview of the watchdog memory map.

Table 21-2. Watchdog Timer Module Memory Map

IPSBAR Offset	[31:24]	[23:16]	[15:8]	[7:0]	Access <sup>1</sup>
0x14_0000	Watchdog Control Register (WCR)		Watchdog Modulus Register (WMR)		S
0x14_0004	Watchdog Count Register (WCNTR)		Watchdog Service Register (WSR)		S/U

<sup>1</sup> S = CPU supervisor mode access only. S/U = CPU supervisor or user mode access. User mode accesses to supervisor only addresses have no effect and result in a cycle termination transfer error.

### 21.2.1 Register Description

The watchdog timer programming model consists of these registers:

- Watchdog control register (WCR), which configures watchdog timer operation
- Watchdog modulus register (WMR), which determines the timer modulus reload value
- Watchdog count register (WCNTR), which provides visibility to the watchdog counter value

- Watchdog service register (WSR), which requires a service sequence to prevent reset

### 21.2.1.1 Watchdog Control Register (WCR)

The 16-bit WCR configures watchdog timer operation.

	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	WAIT	DOZE	HALTED	EN
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1
Address	IPSBAR + 0x14_0000															

**Figure 21-2. Watchdog Control Register (WCR)**

**Table 21-3. WCR Field Descriptions**

Bits	Name	Description
15–4	—	Reserved, should be cleared.
3	WAIT	Wait mode bit. Controls the function of the watchdog timer in wait mode. Once written, the WAIT bit is not affected by further writes except in halted mode. Reset sets WAIT. 0 Watchdog timer not affected in wait mode 1 Watchdog timer stopped in wait mode
2	DOZE	Doze mode bit. Controls the function of the watchdog timer in doze mode. Once written, the DOZE bit is not affected by further writes except in halted mode. Reset sets DOZE. 0 Watchdog timer not affected in doze mode 1 Watchdog timer stopped in doze mode
1	HALTED	Halted mode bit. Controls the function of the watchdog timer in halted mode. Once written, the HALTED bit is not affected by further writes except in halted mode. During halted mode, watchdog timer registers can be written and read normally. When halted mode is exited, timer operation continues from the state it was in before entering halted mode, but any updates made in halted mode remain. If a write-once register is written for the first time in halted mode, the register is still writable when halted mode is exited. 0 Watchdog timer not affected in halted mode 1 Watchdog timer stopped in halted mode Note: Changing the HALTED bit from 1 to 0 during halted mode starts the watchdog timer. Changing the HALTED bit from 0 to 1 during halted mode stops the watchdog timer.
0	EN	Watchdog enable bit. Enables the watchdog timer. Once written, the EN bit is not affected by further writes except in halted mode. When the watchdog timer is disabled, the watchdog counter and prescaler counter are held in a stopped state. 0 Watchdog timer disabled 1 Watchdog timer enabled

### 21.2.1.2 Watchdog Modulus Register (WMR)

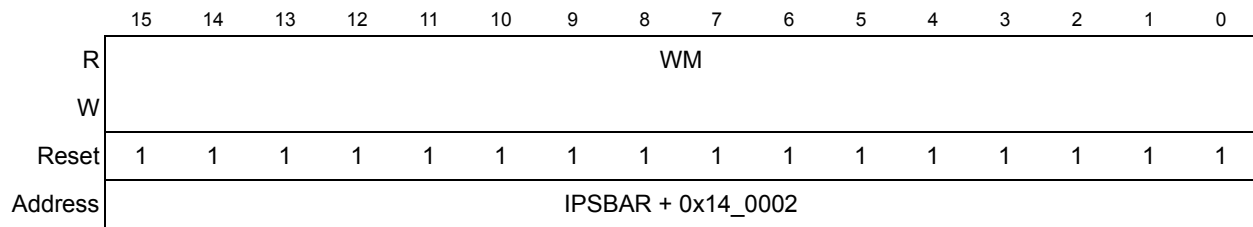


Figure 21-3. Watchdog Modulus Register (WMR)

Table 21-4. WMR Field Descriptions

Bits	Name	Description
15–0	WM	Watchdog modulus. Contains the modulus that is reloaded into the watchdog counter by a service sequence. Once written, the WM[15:0] field is not affected by further writes except in halted mode. Writing to WMR immediately loads the new modulus value into the watchdog counter. The new value is also used at the next and all subsequent reloads. Reading WMR returns the value in the modulus register. Reset initializes the WM[15:0] field to 0xFFFF. Note: The prescaler counter is reset anytime a new value is loaded into the watchdog counter and also during reset.

### 21.2.1.3 Watchdog Count Register (WCNTR)

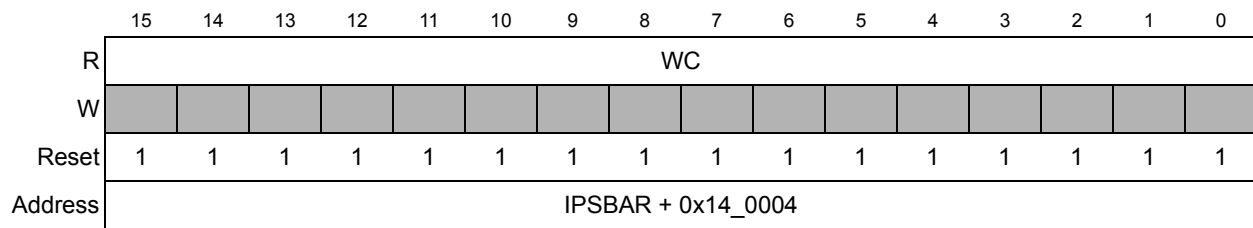


Figure 21-4. Watchdog Count Register (WCNTR)

Table 21-5. WCNTR Field Descriptions

Bits	Name	Description
15–0	WC	Watchdog count field. Reflects the current value in the watchdog counter. Reading the 16-bit WCNTR with two 8-bit reads is not guaranteed to return a coherent value. Writing to WCNTR has no effect, and write cycles are terminated normally.

### 21.2.1.4 Watchdog Service Register (WSR)

When the watchdog timer is enabled, writing 0x5555 and then 0xAAAA to WSR before the watchdog counter times out prevents a reset. If WSR is not serviced before the timeout, the watchdog timer sends a signal to the reset controller module that sets the RSR[WDR] bit and asserts a system reset.



Both writes must occur in the order listed before the timeout, but any number of instructions can be executed between the two writes. However, writing any value other than 0x5555 or 0xAAAA to WSR resets the servicing sequence, requiring both values to be written to keep the watchdog timer from causing a reset.

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	WS															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Address	IPSBAR + 0x0014_0006															

**Figure 21-5. Watchdog Service Register (WSR)**



# Chapter 22

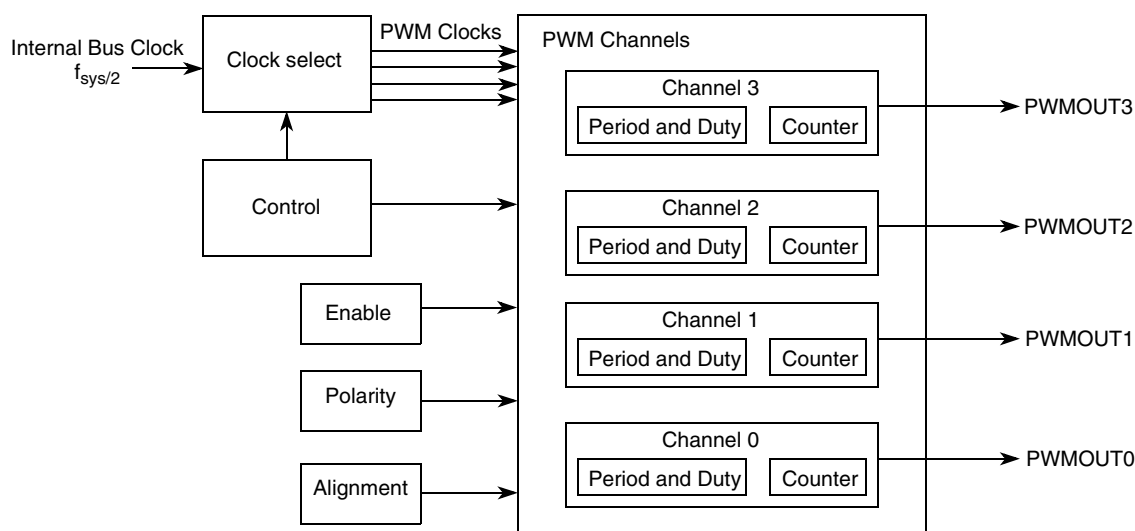
## Pulse Width Modulation (PWM) Module

### 22.1 Introduction

This chapter describes the configuration and operation of the pulse width modulation (PWM) module for the MCF5275 device. It includes a block diagram, programming model, and functional description.

#### 22.1.1 Overview

The PWM module shown in [Figure 22-1](#), generates a synchronous series of pulses having programmable period and duty cycle. With a suitable low-pass filter, the PWM can be used as a digital-to-analog converter.



**Figure 22-1. PWM Block Diagram**

Summary of the main features include:

- Double-buffered period and duty cycle
- Left or center aligned outputs
- Four independent PWM modules
- Byte-wide registers provide programmable duty cycle and period control
- Four programmable clock sources

**NOTE**

The GPIO module must be configured to enable the peripheral function of the appropriate pins (refer to [Chapter 12, “General Purpose I/O Module”](#)) prior to configuring the PWM module.

## 22.2 Memory Map/Register Definition

This section describes the registers and control bits in the PWM module. There are four independent PWM modules, each with its own control and counter registers. The memory map for the PWM is shown in [Table 22-1](#).

**Table 22-1. PWM Module Memory Map**

IPSBAR Offset	[31:24]	[23:16]	[15:8]	[7:0]
0x1D_0000	PWM Enable Register (PWME)	PWM Polarity Register (PWMPOL)	PWM Clock Select Register (PWMCLK)	PWM Prescale Clock Select Register (PWMPRCLK)
0x1D_0004	PWM Center Align Enable Register (PWMCAE)	PWM Control Register (PWMCTL)	Reserved	
0x1D_0008	PWM Scale A Register (PWMSCLA)	PWM Scale B Register (PWMSCLB)	Reserved	
0x1D_000C	PWM 0 Counter Register (PWMCNT0)	PWM 1 Counter Register (PWMCNT1)	PWM 2 Counter Register (PWMCNT2)	PWM 3 Counter Register (PWMCNT3)
0x1D_0010	Reserved		PWM 0 Period Register (PWMPER0)	PWM 1 Period Register (PWMPER1)
0x1D_0014	PWM 2 Period Register (PWMPER2)	PWM 3 Period Register (PWMPER3)	Reserved	
0x1D_0018	PWM 0 Duty Register (PWMDTY0)	PWM 1 Duty Register (PWMDTY1)	PWM 2 Duty Register (PWMDTY2)	PWM 3 Duty Register (PWMDTY3)

### 22.2.1 PWM Enable Register (PWME)

Each PWM channel has an enable bit ( $PWME_n$ ) to start its waveform output. While in run mode, if all four PWM channels are disabled,  $PWME[3:0] = 0$ , the prescaler counter shuts off for power savings. See [Section 22.3.2.1, “PWM Enable”](#) for more information.

	7	6	5	4	3	2	1	0
R	0	0	0	0	PWME3	PWME2	PWME1	PWME0
W								
Reset	0	0	0	0	0	0	0	0
Address	IPSBAR + 0x1D_0000							

**Figure 22-2. PWM Enable Register (PWME)**

**Table 22-2. PWME Field Descriptions**

Bits	Name	Description
7–4	—	Reserved, should be cleared.
3	PWME3	PWM channel 3 output enable. If enabled, the PWM signal becomes available at PWMOUT3 when its corresponding clock source begins its next cycle. 0 PWM output disabled 1 PWM output enabled
2	PWME2	PWM channel 2 output enable. If enabled, the PWM signal becomes available at PWMOUT2 when its corresponding clock source begins its next cycle. If PWMCTL[CON23] is set, then this bit has no effect and PWMOUT2 is disabled. 0 PWM output disabled 1 PWM output enabled, if PWMCTL[CON23]=0
1	PWME1	PWM channel 1 output enable. If enabled, the PWM signal becomes available at PWMOUT1 when its corresponding clock source begins its next cycle. 0 PWM output disabled 1 PWM output enabled
0	PWME0	PWM channel 0 output enable. If enabled, the PWM signal becomes available at PWMOUT0 when its corresponding clock source begins its next cycle. If PWMCTL[CON01] is set, then this bit has no effect and PWMOUT0 is disabled. 0 PWM output disabled 1 PWM output enabled, if PWMCTL[CON01]=0

## 22.2.2 PWM Polarity Register (PWMPOL)

The starting polarity of each PWM channel waveform is determined by the associated PWMPOL[PPOL $n$ ] bit. If the polarity is changed while a PWM signal is being generated, a truncated or stretched pulse can occur during the transition.

	7	6	5	4	3	2	1	0
R	0	0	0	0	PPOL3	PPOL2	PPOL1	PPOL0
W								
Reset	0	0	0	0	0	0	0	0
Address	IPSBAR + 0x1D_0001							

**Figure 22-3. PWM Polarity Register (PWMPOL)****Table 22-3. PWMPOL Field Descriptions**

Bits	Name	Description
7–4	—	Reserved, should be cleared.
3–0	PPOL $n$	PWM channel $n$ polarity. 0 PWM channel $n$ output is low at the beginning of the period, then goes high when the duty count is reached 1 PWM channel $n$ output is high at the beginning of the period, then goes low when the duty count is reached

## 22.2.3 PWM Clock Select Register (PWMCLK)

Each PWM channel has the capability of selecting one of two clocks. For channels 0 and 1 the clock choices are clock A or SA. For channels 2 and 3 the choices are clock B or SB. The clock selection is done with the below PWMCLK[PCLKx] control bits. If a clock select is changed while a PWM signal is being generated, a truncated or stretched pulse can occur during the transition.

	7	6	5	4	3	2	1	0
R	0	0	0	0	PCLK3	PCLK2	PCLK1	PCLK0
W								
Reset	0	0	0	0	0	0	0	0
Address	IPSBAR + 0x1D_0002							

**Figure 22-4. PWM Clock Select Register (PWMCLK)**

**Table 22-4. PWMCLK Field Descriptions**

Bits	Name	Description															
7–4	—	Reserved, should be cleared.															
3–0	PCLK $n$	<p>PWM channel <math>n</math> clock select. Selects between one of two clock sources for each PWM channel. See <a href="#">Section 22.2.4, “PWM Prescale Clock Select Register (PWMPRCLK)”</a> and <a href="#">Section 22.2.7, “PWM Scale A Register (PWMSCLA)”</a> for more information on how the different clock rates are generated.</p> <table><tr><th></th><th>PCLK3 (PWM3 Clock Source)</th><th>PCLK2 (PWM2 Clock Source)</th><th>PCLK1 (PWM1 Clock Source)</th><th>PCLK0 (PWM0 Clock Source)</th></tr><tr><td>0</td><td>B</td><td>B</td><td>A</td><td>A</td></tr><tr><td>1</td><td>SB</td><td>SB</td><td>SA</td><td>SA</td></tr></table>		PCLK3 (PWM3 Clock Source)	PCLK2 (PWM2 Clock Source)	PCLK1 (PWM1 Clock Source)	PCLK0 (PWM0 Clock Source)	0	B	B	A	A	1	SB	SB	SA	SA
	PCLK3 (PWM3 Clock Source)	PCLK2 (PWM2 Clock Source)	PCLK1 (PWM1 Clock Source)	PCLK0 (PWM0 Clock Source)													
0	B	B	A	A													
1	SB	SB	SA	SA													

## 22.2.4 PWM Prescale Clock Select Register (PWMPRCLK)

The PWMPRCLK register selects the prescale clock source for clocks A and B independently. If the clock prescale is changed while a PWM signal is being generated, a truncated or stretched pulse can occur during the transition.

	7	6	5	4	3	2	1	0
R	0	PCKB			0	PCKA		
W								
Reset	0	0	0	0	0	0	0	0
Address	IPSBAR + 0x1D_0003							

**Figure 22-5. PWM Prescale Clock Select Register (PWMPRCLK)**

**Table 22-5. PWMPRCLK Field Descriptions**

Bits	Name	Description										
7	—	Reserved, should be cleared.										
6–4	PCKB	Clock B prescaler select. These three bits control the rate of Clock B which can be used for PWM channels 2 and 3. <table><tr><th>PCKB</th><th>Clock B Rate</th></tr><tr><td>000</td><td>Internal bus clock <math>\div 2^0</math></td></tr><tr><td>001</td><td>Internal bus clock <math>\div 2^1</math></td></tr><tr><td>...</td><td>...</td></tr><tr><td>111</td><td>Internal bus clock <math>\div 2^7</math></td></tr></table>	PCKB	Clock B Rate	000	Internal bus clock $\div 2^0$	001	Internal bus clock $\div 2^1$	...	...	111	Internal bus clock $\div 2^7$
PCKB	Clock B Rate											
000	Internal bus clock $\div 2^0$											
001	Internal bus clock $\div 2^1$											
...	...											
111	Internal bus clock $\div 2^7$											
3	—	Reserved, should be cleared.										
2–0	PCKA	Clock A prescaler select. These three bits control the rate of Clock A which can be used for PWM channels 0 and 1. <table><tr><th>PCKA</th><th>Clock A Rate</th></tr><tr><td>000</td><td>Internal bus clock <math>\div 2^0</math></td></tr><tr><td>001</td><td>Internal bus clock <math>\div 2^1</math></td></tr><tr><td>...</td><td>...</td></tr><tr><td>111</td><td>Internal bus clock <math>\div 2^7</math></td></tr></table>	PCKA	Clock A Rate	000	Internal bus clock $\div 2^0$	001	Internal bus clock $\div 2^1$	...	...	111	Internal bus clock $\div 2^7$
PCKA	Clock A Rate											
000	Internal bus clock $\div 2^0$											
001	Internal bus clock $\div 2^1$											
...	...											
111	Internal bus clock $\div 2^7$											

### 22.2.5 PWM Center Align Enable Register (PWMCAE)

The PWMCAE register contains four control bits for the selection of center aligned outputs or left aligned outputs for each PWM channel. Write these bits only when the corresponding channel is disabled. See [Section 22.3.2.5, “Left Aligned Outputs”](#) and [Section 22.3.2.6, “Center Aligned Outputs”](#) for a more detailed description of the PWM output modes.

	7	6	5	4	3	2	1	0
R	0	0	0	0	CAE3	CAE2	CAE1	CAE0
W								
Reset	0	0	0	0	0	0	0	0
Address	IPSBAR + 0x1D_0004							

**Figure 22-6. PWM Center Align Enable Register (PWMCAE)**

**Table 22-6. PWMCAE Field Descriptions**

Bits	Name	Description
7–4	—	Reserved, should be cleared.
3–0	CAEn	Center align enable for channel <i>n</i> . 0 Channel <i>n</i> operates in left aligned output mode 1 Channel <i>n</i> operates in center aligned output mode

## 22.2.6 PWM Control Register (PWMCTL)

The PWMCTL register provides various control of the PWM module. Change the CON<sub>nn</sub> bits only when both corresponding channels are disabled. See [Section 22.3.2.7, “PWM 16-Bit Functions”](#) for a more detailed description of the concatenation function.

	7	6	5	4	3	2	1	0
R	0	0	CON23	CON01	PSWAI	PFRZ	0	0
W								
Reset	0	0	0	0	0	0	0	0
Address	IPSBAR + 0x1D_0005							

**Figure 22-7. PWM Control Register (PWMCTL)****Table 22-7. PWMCTL Field Descriptions**

Bits	Name	Description
7–6	—	Reserved, should be cleared.
5	CON23	Concatenates PWM channels 2 and 3 to form one 16-bit PWM channel. 0 Channels 2 and 3 are separate 8-bit PWMs 1 Concatenate PWM 2 and 3. Channel 2 becomes the high order byte and channel 3 the low order byte. PWMOUT3 is the output for this 16-bit PWM signal, and PWMOUT2 is disabled. The channel 3 clock select, polarity, center align enable, and enable bits control this concatenated output.
4	CON01	Concatenates PWM channels 0 and 1 to form one 16-bit PWM channel. 0 Channels 0 and 1 are separate 8-bit PWMs 1 Concatenate PWM 0 and 1. Channel 0 becomes the high order byte and channel 1 the low order byte. PWMOUT1 is the output for this 16-bit PWM signal, and PWMOUT0 is disabled. The channel 1 clock select, polarity, center align enable, and enable bits control this concatenated output.
3	PSWAI	PWM stops in wait mode. Disables the input clock to the prescaler while in wait mode. 0 Allow the clock to the prescaler while in wait mode 1 Stop the input clock to the prescaler whenever the core is in wait mode
2	PFRZ	PWM counters stop in freeze mode. 0 Allow PWM counters to continue while in freeze mode 1 Disable PWM input clock to the prescaler when the core is in freeze mode. Useful for emulation as it allows the PWM function to be suspended.
1–0	—	Reserved, should be cleared.

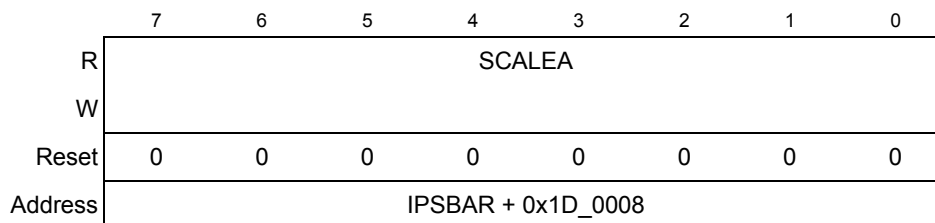


## 22.2.7 PWM Scale A Register (PWMSCLA)

PWMSCLA is the programmable scale value used in scaling clock A to generate clock SA. Clock SA is generated according to the following equation:

$$\text{Clock SA} = \frac{\text{Clock A}}{2 \times \text{PWMSCLA}}$$

Any value written to this register will cause the scale counter to load the new scale value (PWMSCLA).



**Figure 22-8. PWM Scale A Register (PWMSCLA)**

**Table 22-8. PWMSCLA Field Descriptions**

Bits	Name	Description												
7–0	SCALEA	Part of divisor used to form Clock SA from Clock A. <table><tr><th>SCALEA</th><th>Value</th></tr><tr><td>0x00</td><td>256</td></tr><tr><td>0x01</td><td>1</td></tr><tr><td>0x02</td><td>2</td></tr><tr><td>...</td><td>...</td></tr><tr><td>0xFF</td><td>255</td></tr></table>	SCALEA	Value	0x00	256	0x01	1	0x02	2	...	...	0xFF	255
SCALEA	Value													
0x00	256													
0x01	1													
0x02	2													
...	...													
0xFF	255													

## 22.2.8 PWM Scale B Register (PWMSCLB)

PWMSCLB is the programmable scale value used in scaling clock B to generate clock SB. Clock SB is generated according to the following equation:

$$\text{Clock SB} = \frac{\text{Clock B}}{2 \times \text{PWMSCLB}}$$

Any value written to this register will cause the scale counter to load the new scale value (PWMSCLB).



Figure 22-9. PWM Scale B Register (PWMSCLB)

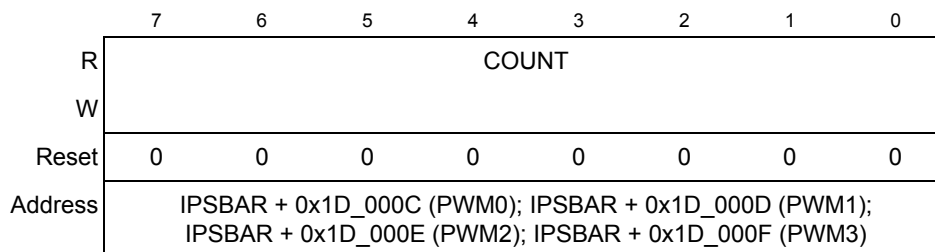
Table 22-9. PWMSCLB Field Descriptions

Bits	Name	Description												
7–0	SCALEB	Divisor used to form Clock SB from Clock B. <table><tr><th>SCALEB</th><th>Value</th></tr><tr><td>0x00</td><td>256</td></tr><tr><td>0x01</td><td>1</td></tr><tr><td>0x02</td><td>2</td></tr><tr><td>...</td><td>...</td></tr><tr><td>0xFF</td><td>255</td></tr></table>	SCALEB	Value	0x00	256	0x01	1	0x02	2	...	...	0xFF	255
SCALEB	Value													
0x00	256													
0x01	1													
0x02	2													
...	...													
0xFF	255													

22.2.9 PWM Channel Counter Registers (PWMCNTn)

Each channel has a dedicated 8-bit up/down counter which runs at the rate of the selected clock source, PWMCLK[PCLKn]. The user can read the counters at any time without affecting the count or the operation of the PWM channel. Any value written to the counter causes the counter to reset to 0x00, the counter direction to be set to up for center-aligned mode, the immediate load of both duty and period registers with values from the buffers, and the output to change according to the polarity bit.

The counter is also cleared at the end of the effective period (see [Section 22.3.2.5, “Left Aligned Outputs”](#) and [Section 22.3.2.6, “Center Aligned Outputs”](#) for more details). When the channel is disabled (PWME<sub>n</sub>=0), the PWMCNT<sub>n</sub> register does not count. When a channel is enabled (PWME<sub>n</sub>=1), the associated PWM counter starts at the count in the PWMCNT<sub>n</sub> register. For more detailed information on the operation of the counters, refer to [Section 22.3.2.4, “PWM Timer Counters.”](#)

**Figure 22-10. PWM Counter Registers (PWMCNT $n$ )****Table 22-10. PWMCNT $n$  Field Descriptions**

Bits	Name	Description
7–0	COUNT	Current value of the PWM up counter. Reset to zero when written.

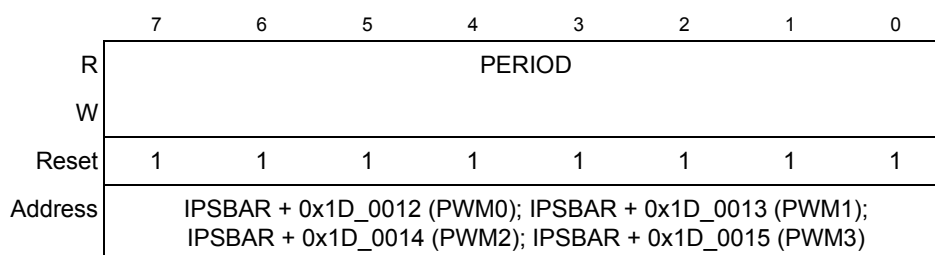
## 22.2.10 PWM Channel Period Registers (PWMPER $n$ )

The PWM period registers determine the period of the associated PWM channel. Refer to [Section 22.3.2.3, “PWM Period and Duty”](#) for more information.

Calculating the output period depends on the output mode (center aligned has twice the period as left aligned mode) as well as PWMPER $n$ . See the below equation:

$$\text{PWM } n \text{ period} = \text{Channel clock period} \times (\text{PWMCAE}[\text{CAEn}] + 1) \times \text{PWMPER}_n$$

For Boundary Case programming values (e.g. PWMPER $n$  = 0x00), please refer to [Section 22.3.2.8, “PWM Boundary Cases.”](#)

**Figure 22-11. PWM Period Registers (PWMPER $n$ )****Table 22-11. PWMPER $n$  Field Descriptions**

Bits	Name	Description
7–0	PERIOD	Period counter for the output PWM signal. If PERIOD = 0x00, the PWM $n$ output is always high (PPOL $n$ =1) or always low (PPOL $n$ =0). See <a href="#">Section 22.3.2.8, “PWM Boundary Cases”</a> for other special cases.

## 22.2.11 PWM Channel Duty Registers (PWMDTY<sub>n</sub>)

The PWM duty registers determine the duty cycle of the associated PWM channel. To calculate the output duty cycle (high time as a percentage of period) for a particular channel:

$$\text{Duty Cycle} = \left| \left( 1 - \text{PWMPOL}[\text{PPOL}_n] - \frac{\text{PWMDTY}_n}{\text{PWMPER}_n} \right) \right| \times 100\%$$

For Boundary Case programming values (e.g.  $\text{PWMDTY}_n = 0x00$  or  $\text{PWMDTY}_n > \text{PWMPER}_n$ ), please refer to Section [Section 22.3.2.8, “PWM Boundary Cases”](#).

	7	6	5	4	3	2	1	0
R	DUTY							
W								
Reset	1	1	1	1	1	1	1	1
Address	IPSBAR + 0x1D_0018 (PWM0); IPSBAR + 0x1D_0019 (PWM1); IPSBAR + 0x1D_001A (PWM2); IPSBAR + 0x1D_001B (PWM3)							

**Figure 22-12. PWM Duty Registers (PWMDTY<sub>n</sub>)**

**Table 22-12. PWMDTY<sub>n</sub> Field Descriptions**

Bits	Name	Description
7–0	DUTY	Contains the duty value used to determine when a transition will occur on the PWM output signal. When a match occurs with the corresponding PWMCNT <sub>n</sub> register, the PWM output will toggle. If DUTY = 0x00, the PWM <sub>n</sub> output is always low (PPOL <sub>n</sub> =1) or always high (PPOL <sub>n</sub> =0). See <a href="#">Section 22.3.2.8, “PWM Boundary Cases”</a> for other special cases.

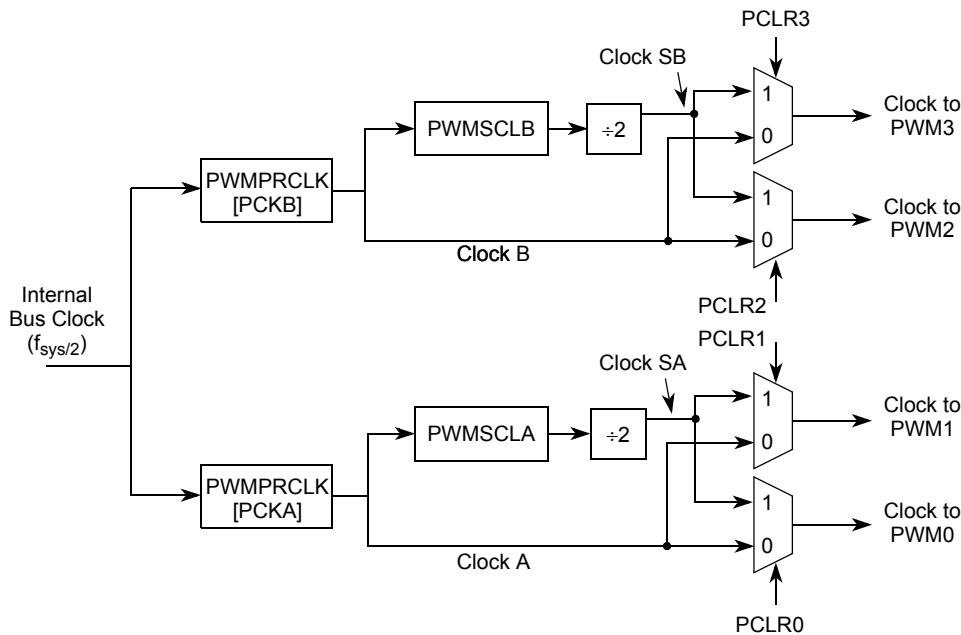
## 22.3 Functional Description

### 22.3.1 PWM Clock Select

There are four available clocks named Clock A, B, SA (Scaled A), and SB (Scaled B), all of which are based on the internal bus clock.

Clock A and B can be programmed to run at 1, 1/2, ..., 1/128 times the internal bus clock. Clock SA and SB use clock A and B respectively as an input and divides it further with a reloadable counter. The rates available for clock SA and SB are programmable to run at clock A and B divided by 2, 4, ..., or 512. Each PWM channel has the capability of selecting one of two clocks,

either the prescaled clock (clock A or B) or the scaled clock (clock SA or SB). The block diagram in Figure 22-13 shows the four different clocks and how the scaled clocks are created.



**Figure 22-13. PWM Clock Select Block Diagram**

### 22.3.1.1 Prescaled Clock (A or B)

The internal bus clock is the input clock to the PWM prescaler which can be disabled when the device is in Freeze mode by setting the PWMCTL[PFRZ] bit. This is useful for reducing power consumption and for emulation in order to freeze the PWM. The input clock is also disabled when all four PWM channels are disabled (PWME<sub>n</sub>=0).

Clock A and B are scaled values of the input clock. The value is software selectable for both clock A and B and has options of 1, 1/2, ..., or 1/128 times the internal bus clock. The value selected for clock A and B are determined by the PWMPRCLK[PCKA<sub>n</sub>] and PWMPRCLK[PCKB<sub>n</sub>] bits.

### 22.3.1.2 Scaled Clock (SA or SB)

The scaled A (SA) and scaled B (SB) clocks use clock A and B respectively as inputs and divide it further with a user programmable value and then divide this by 2. The rates available for clock SA are programmable to run at clock A divided by 2, 4, ..., or 512. Similar rates are available for clock SB.

Clock SA equals Clock A divided by two times the value in the PWMSCLA register:

$$\text{Clock SA} = \frac{\text{Clock A}}{2 \times \text{PWMSCLA}}$$

Similarly, Clock SB is generated according to the following equation:

$$\text{Clock SB} = \frac{\text{Clock B}}{2 \times \text{PWMSCLB}}$$

As an example, consider the case in which the user writes 0xFF into the PWMSCLA register. Clock A for this case is selected to be internal bus clock divided by 4. A pulse will occur at a rate of once every  $255 \times 4$  bus cycles. Passing this through the divide by two circuit produces a clock signal of the internal bus clock divided by 2040. Similarly, a value of 0x01 in the PWMSCLA register when Clock A is internal bus clock divided by 4 will produce an internal bus clock divided by 8 rate.

Writing to PWMSCLA or PWMSCLB causes the associated 8-bit down counter to be re-loaded. Otherwise, when changing rates the counter would have to count down to 0x01 before counting at the proper rate. Forcing the associated counter to re-load the scale register value every time PWMSCLA or PWMSCLB is written prevents this.

Writing to the scale registers while channels are operating can cause irregularities in the PWM outputs.

### 22.3.1.3 Clock Select

Each PWM channel has the capability of selecting one of two clocks. For channels 0 and 1 the clock choices are clock A or SA. For channels 2 and 3 the choices are clock B or SB. The clock selection is done with the PWMCLK[PCLKx] control bits.

Changing clock control bits while channels are operating can cause irregularities in the PWM outputs.

## 22.3.2 PWM Channel Timers

The main part of the PWM module is the actual timers. Each of the timer channels has a counter, a period register and a duty register (each are 8-bit). The waveform output period is controlled by a match between the period register and the value in the counter. The duty is controlled by a match between the duty register and the counter value and causes the state of the output to change during the period. The starting polarity of the output is also selectable on a per channel basis. [Figure 22-14](#) shows a block diagram for PWM timer.

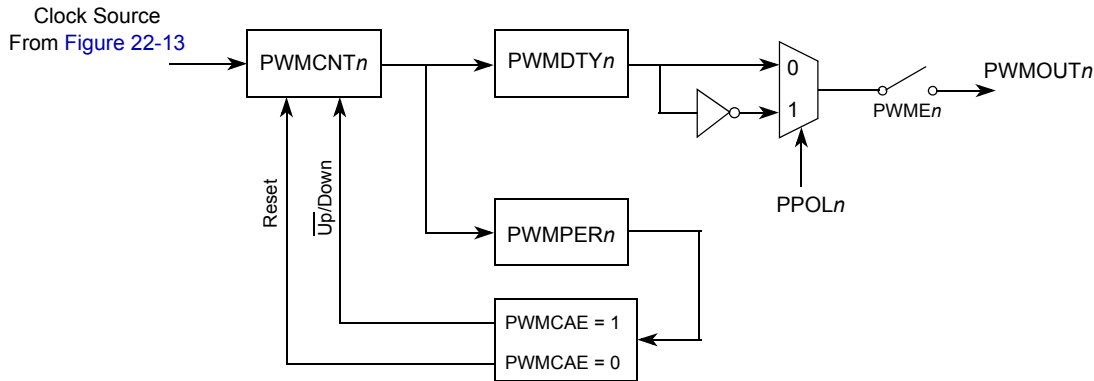


Figure 22-14. PWM Timer Channel Block Diagram

### 22.3.2.1 PWM Enable

Each PWM channel has an enable bit ( $PWME_n$ ) to start its waveform output. When any of the  $PWME_n$  bits are set ( $PWME_n=1$ ), the associated PWM output signal is enabled immediately. However, the actual PWM waveform is not available on the associated PWM output until its clock source begins its next cycle due to the synchronization of  $PWME_n$  and the clock source. An exception to this is when channels are concatenated. Refer to [Section 22.3.2.7, “PWM 16-Bit Functions”](#) for more detail.

Note that the first PWM cycle after enabling the channel can be irregular. When the channel is disabled ( $PWME_n=0$ ), the counter for the channel does not count.

### 22.3.2.2 PWM Polarity

Each channel has a polarity bit to allow starting a waveform cycle with a high or low signal. This is shown on the block diagram as a mux select. When one of the bits in the  $PWMPOL$  register is set, the associated PWM channel output is high at the beginning of the waveform, then goes low when the duty count is reached. Conversely, if the polarity bit is zero, the output starts low and then goes high when the duty count is reached.

### 22.3.2.3 PWM Period and Duty

Dedicated period and duty registers exist for each channel and are double buffered so that if they change while the channel is enabled, the change will not take effect until one of the following occurs:

- The effective period ends
- The  $PWMCNT_n$  register is written (counter resets to 0x00)
- The channel is disabled,  $PWME_n = 0$

In this way, the output of the PWM will always be either the old waveform or the new waveform, not some variation in between. If the channel is not enabled, then writes to the period and duty registers will go directly to the latches as well as the buffer.

A change in duty or period can be forced into effect immediately by writing the new value to the duty and/or period registers and then writing to the counter. This forces the counter to reset and the new duty and/or period values to be latched. In addition, since the counter is readable it is possible to know where the count is with respect to the duty value and software can be used to make adjustments. When forcing a new period or duty into effect immediately, an irregular PWM cycle can occur.

Depending on the polarity bit, the duty registers will contain the count of either the high time or the low time.

#### 22.3.2.4 PWM Timer Counters

Each channel has a dedicated 8-bit up/down counter which runs at the rate of the selected clock source (see [Figure 22-13](#) for the available clock sources and rates). The counter compares to two registers, a duty register and a period register as shown in [Figure 22-14](#). When the PWM counter matches the duty register the output flip-flop changes state causing the PWM waveform to also change state. A match between the PWM counter and the period register behaves differently depending on what output mode is selected as shown in [Figure 22-14](#) and described in [Section 22.3.2.5, “Left Aligned Outputs”](#) and [Section 22.3.2.6, “Center Aligned Outputs”](#).

Each channel counter can be read at anytime without affecting the count or the operation of the PWM channel.

Any value written to the counter causes the counter to reset to 0x00, the counter direction to be set to up, the immediate load of both duty and period registers with values from the buffers, and the output to change according to the polarity bit. When the channel is disabled ( $PWME_n = 0$ ), the counter stops. When a channel becomes enabled ( $PWME_n = 1$ ), the associated PWM counter continues from the count in the  $PWMCNT_n$  register. This allows the waveform to continue where it left off when the channel is re-enabled. When the channel is disabled, writing “0” to the period register will cause the counter to reset on the next selected clock.

#### NOTE

If the user wants to start a new “clean” PWM waveform without any “history” from the old waveform, the user must write to channel counter ( $PWMCNT_n$ ) prior to enabling the PWM channel ( $PWME_n = 1$ ).

Generally, writes to the counter are done prior to enabling a channel in order to start from a known state. However, writing a counter can also be done while the PWM channel is enabled (counting). The effect is similar to writing the counter when the channel is disabled except that the new period



is started immediately with the output set according to the polarity bit. Writing to the counter while the channel is enabled can cause an irregular PWM cycle to occur.

The counter is cleared at the end of the effective period (see [Section 22.3.2.5, “Left Aligned Outputs”](#) and [Section 22.3.2.6, “Center Aligned Outputs”](#) for more details).

**Table 22-13. PWM Timer Counter Conditions**

Counter Clears (0x00)	Counter Counts	Counter Stops
When PWMCNT $n$ register written to any value	When PWM channel is enabled (PWME $n$ = 1). Counts from last value in PWMCNT $n$ .	When PWM channel is disabled (PWME $n$ = 0)
Effective period ends		

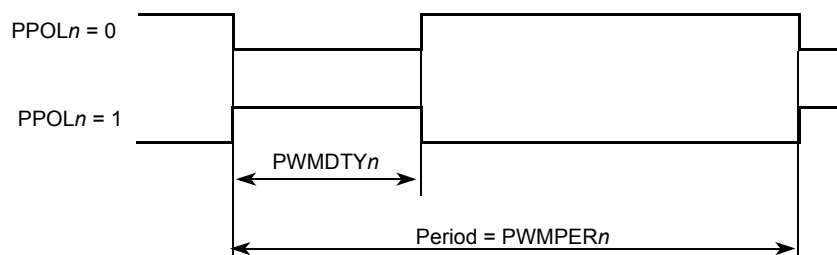
### 22.3.2.5 Left Aligned Outputs

The PWM timer provides the choice of two types of outputs, Left Aligned or Center Aligned outputs. They are selected with the PWMCAE[CA $n$ ] bits. If the CA $n$  bit is cleared, the corresponding PWM output will be left aligned.

In left aligned output mode, the 8-bit counter is configured as an up counter only. It compares to two registers, a duty register and a period register as shown in the block diagram in [Figure 22-14](#). When the PWM counter matches the duty register the output flip-flop changes state causing the PWM waveform to also change state. A match between the PWM counter and the period register resets the counter and the output flip-flop as shown in [Figure 22-14](#) as well as performing a load from the double buffer period and duty register to the associated registers as described in [Figure 22.3.2.3](#). The counter counts from 0 to the value in the period register minus 1.

#### NOTE

Changing the PWM output mode from left aligned output to center aligned output (or vice versa) while channels are operating can cause irregularities in the PWM output. It is recommended to program the output mode before enabling the PWM channel.



**Figure 22-15. PWM Left Aligned Output Waveform**

To calculate the output frequency in left aligned output mode for a particular channel, take the selected clock source frequency for the channel (A, B, SA, or SB) and divide it by the value in the period register for that channel.

$$\text{PWMn frequency} = \frac{\text{Clock (A, B, SA, or SB)}}{\text{PWMPERn}}$$

The PWMn duty cycle (high time as a percentage of period) is expressed as:

$$\text{Duty Cycle} = \left( 1 - \text{PWMPOL}[\text{PPOLn}] - \frac{\text{PWMDTYn}}{\text{PWMPERn}} \right) \times 100\%$$

### 22.3.2.5.1 Left Aligned Output Example

As an example of a left aligned output, consider the following case:

Clock Source = internal bus clock, where internal bus clock = 75MHz (13.33ns period)

PPOLn = 0, PWMPERn = 4, PWMDTYn = 1

PWMn Frequency = 75MHz ÷ 4 = 18.75MHz

PWMn Period = 53.33ns

PWMn Duty Cycle =  $\left( 1 - \frac{1}{4} \right) \times 100\% = 75\%$

Shown below is the output waveform generated:

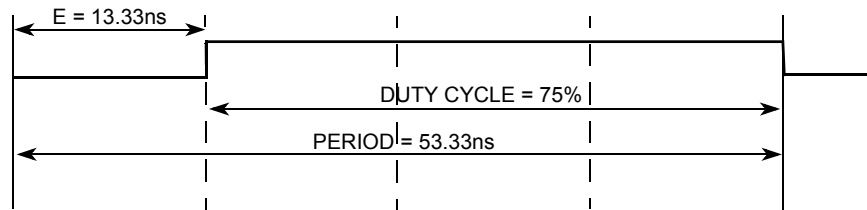


Figure 22-16. PWM Left Aligned Output Example Waveform

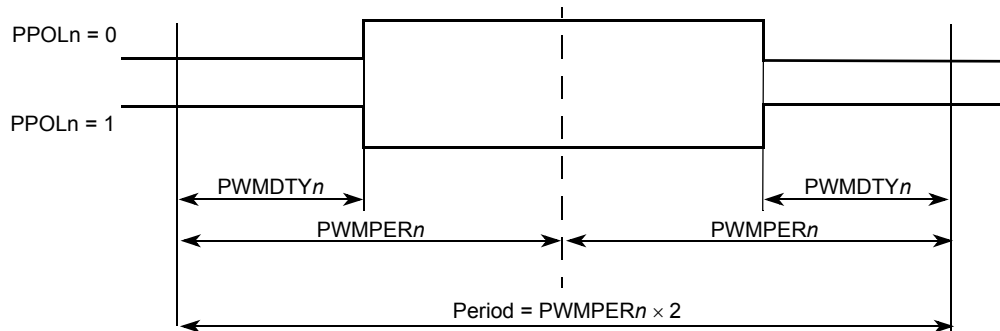
### 22.3.2.6 Center Aligned Outputs

For center aligned output mode selection, set the PWMCAE[CAEn] bit and the corresponding PWM output will be center aligned.

The 8-bit counter operates as an up/down counter in this mode and is set to up whenever the counter is equal to 0x00. The counter compares to two registers, a duty register and a period register as shown in the block diagram in [Figure 22-14](#). When the PWM counter matches the duty register the output flip-flop changes state causing the PWM waveform to also change state. A match between the PWM counter and the period register changes the counter direction from an up-count to a down-count. When the PWM counter decrements and matches the duty register again, the output flip-flop changes state causing the PWM output to also change state. When the

PWM counter decrements and reaches zero, the counter direction changes from a down-count back to an up-count and a load from the double buffer period and duty registers to the associated registers is performed as described in Figure 22.3.2.3. The counter counts from 0 up to the value in the period register and then back down to 0. Thus the effective period is  $PWMPER_n \times 2$ .

Changing the PWM output mode from left aligned output to center aligned output (or vice versa) while channels are operating can cause irregularities in the PWM output. It is recommended to program the output mode before enabling the PWM channel.



**Figure 22-17. PWM Center Aligned Output Waveform**

To calculate the output frequency in center aligned output mode for a particular channel, take the selected clock source frequency for the channel (A, B, SA, or SB) and divide it by twice the value in the period register for that channel.

$$\text{PWM}_n \text{ frequency} = \frac{\text{Clock (A, B, SA, or SB)}}{2 \times PWMPER_n}$$

The  $\text{PWM}_n$  duty cycle (high time as a percentage of period) is expressed as:

$$\text{Duty Cycle} = \left( 1 - \text{PWMPOL}[PPOL_n] - \frac{PWMDTY_n}{PWMPER_n} \right) \times 100\%$$

#### 22.3.2.6.1 Center Aligned Output Example

As an example of a center aligned output, consider the following case:

Clock Source = internal bus clock, where internal bus clock=75MHz (13.3ns period)

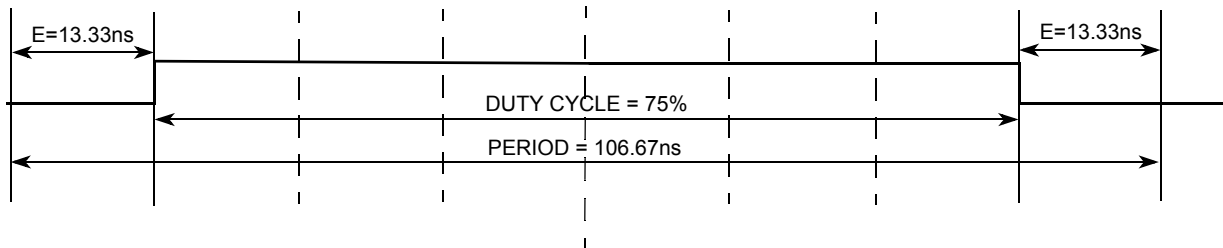
$PPOL_n = 0$ ,  $PWMPER_n = 4$ ,  $PWMDTY_n = 1$

$\text{PWM}_n$  Frequency =  $75\text{MHz}/(2 \times 4) = 9.38\text{MHz}$

$\text{PWM}_n$  Period = 106.67ns

$$\text{PWMn Duty Cycle} = \left(1 - \frac{1}{4}\right) \times 100\% = 75\%$$

Shown below is the output waveform generated.



**Figure 22-18. PWM Center Aligned Output Example Waveform**

### 22.3.2.7 PWM 16-Bit Functions

The PWM timer also has the option of generating 4-channels of 8-bits or 2-channels of 16-bits for greater PWM resolution. This 16-bit channel option is achieved through the concatenation of two 8-bit channels.

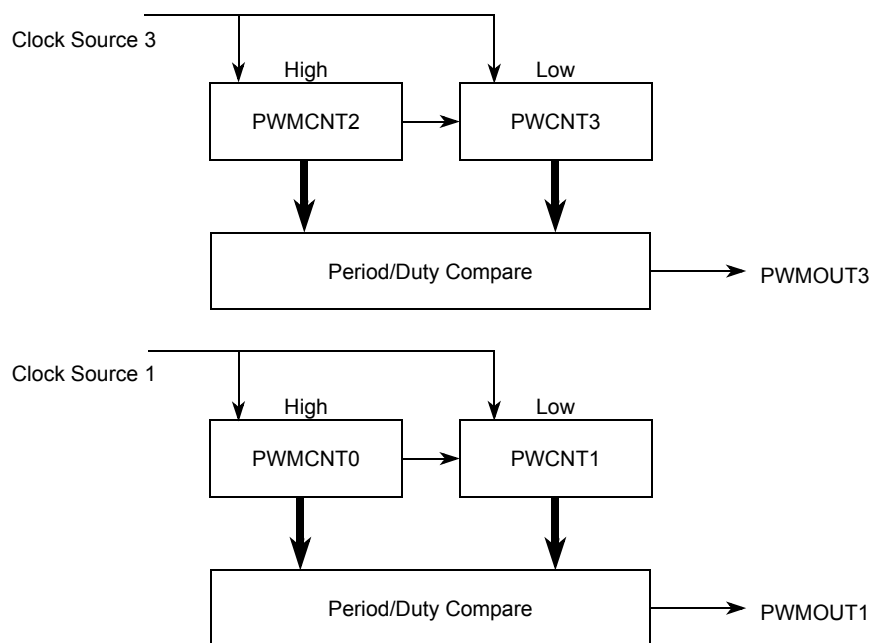
The PWMCTL register contains two concatenation control bits, each of which is used to concatenate a pair of PWM channels into one 16-bit channel. Channels 2 and 3 are concatenated with the CON23 bit, and channels 0 and 1 are concatenated with the CON01 bit. Change these bits only when both corresponding channels are disabled.

As shown in [Figure 22-19](#), when channels 2 and 3 are concatenated, channel 2 registers become the high order bytes of the double byte channel. When channels 0 and 1 are concatenated, channel 0 registers become the high order bytes of the double byte channel.

When using the 16-bit concatenated mode, the clock source is determined by the low order 8-bit channel clock select control bits. That is channel 3 when channels 2 and 3 are concatenated, and channel 1 when channels 0 and 1 are concatenated. The resulting PWM is output to the pins of the corresponding low order 8-bit channel as also shown in [Figure 22-19](#). The polarity of the resulting PWM output is controlled by the PPOL $n$  bit of the corresponding low order 8-bit channel as well.

Once concatenated mode is enabled (PWMCTL[CON $nn$ ] bits set) then enabling/disabling the corresponding 16-bit PWM channel is controlled by the low order PWME $n$  bit. In this case, the high order bytes PWME $n$  bits have no effect and their corresponding PWM output is disabled.

In concatenated mode, writes to the 16-bit counter by using a 16-bit access or writes to either the low or high order byte of the counter will reset the 16-bit counter. Reads of the 16-bit counter must be made by 16-bit access to maintain data coherency.



**Figure 22-19. PWM 16-Bit Mode**

Either left aligned or center aligned output mode can be used in concatenated mode and is controlled by the low order  $CAEn$  bit. The high order  $CAEn$  bit has no effect.

The table shown below is used to summarize which channels are used to set the various control bits when in 16-bit mode.

**Table 22-14. 16-bit Concatenation Mode Summary**

$CONnn$	$PWME_n$	$PPOL_n$	$PCLK_n$	$CAE_n$	$PWM_n$ Output
CON23	PWME3	PPOL3	PCLK3	CAE3	PWMOUT3
CON01	PWME1	PPOL1	PCLK1	CAE1	PWMOUT1

### 22.3.2.8 PWM Boundary Cases

The following table summarizes the boundary conditions for the PWM regardless of the output mode (Left Aligned or Center Aligned) and 8-bit (normal) or 16-bit (concatenation):

**Table 22-15. PWM Boundary Cases**

$PWMDTY_n$	$PWMPER_n$	$PPOL_n$	$PWM_n$ Output
0x00 (indicates no duty)	>0x00	1	Always Low
0x00 (indicates no duty)	>0x00	0	Always High

**Table 22-15. PWM Boundary Cases**

<b>PWMDTY<math>n</math></b>	<b>PWMPER<math>n</math></b>	<b>PPOL<math>n</math></b>	<b>PWM<math>n</math> Output</b>
XX	0x00 <sup>1</sup> (indicates no period)	1	Always High
XX	0x00 <sup>1</sup> (indicates no period)	0	Always Low
$\geq$ PWMPER $n$	XX	1	Always High
$\geq$ PWMPER $n$	XX	0	Always Low

<sup>1</sup>Counter=0x00 and does not count.

# Chapter 23

## Programmable Interrupt Timer Modules (PIT0–PIT3)

### 23.1 Introduction

This chapter describes the operation of the four programmable interrupt timer modules, PIT0–PIT3.

#### 23.1.1 Overview

The PIT is a 16-bit timer that provides precise interrupts at regular intervals with minimal processor intervention. The timer can either count down from the value written in the modulus register, or it can be a free-running down-counter.

#### NOTE

The GPIO module must be configured to enable the peripheral function of the appropriate pins (refer to [Chapter 12, “General Purpose I/O Module”](#)) prior to configuring the PIT modules.

#### 23.1.2 Block Diagram

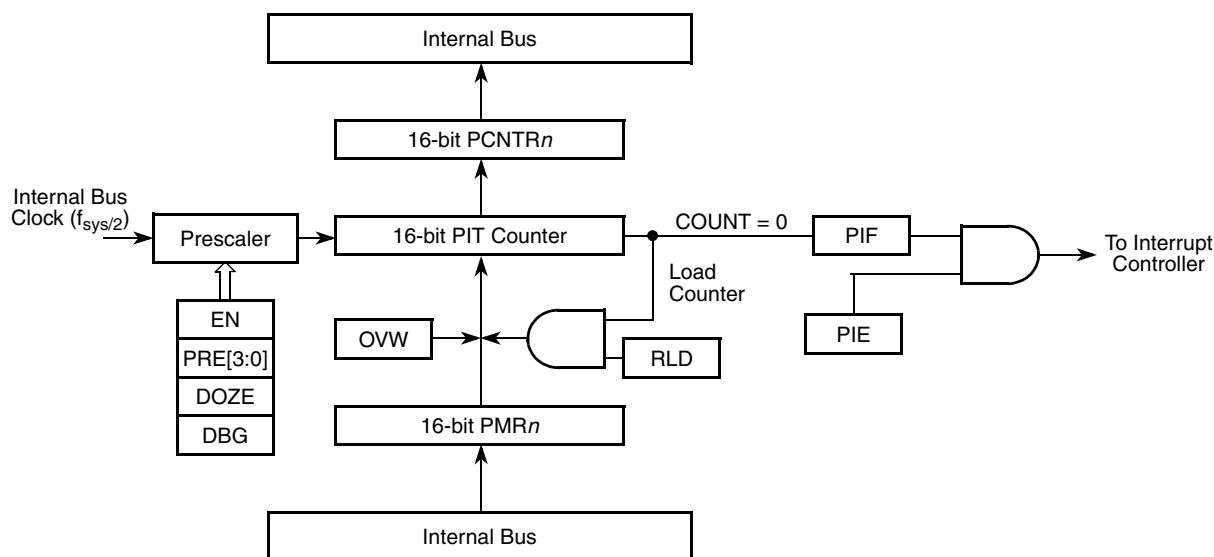


Figure 23-1. PIT Block Diagram

### 23.1.3 Low-Power Mode Operation

This subsection describes the operation of the PIT modules in low-power modes and debug mode of operation. Low-power modes are described in the Power Management Module. [Table 23-1](#) shows the PIT module operation in low-power modes, and how it can exit from each mode.

#### NOTE

The low-power interrupt control register (LPICR) in the System Control Module specifies the interrupt level at or above which the device can be brought out of a low-power mode.

**Table 23-1. PIT Module Operation in Low-power Modes**

Low-power Mode	PIT Operation	Mode Exit
Wait	Normal	N/A
Doze	Normal if PCSR $_n$ [DOZE] cleared, stopped otherwise	Any $\overline{IRQ}_n$ Interrupt at or above level in LPICR
Stop	Stopped	No
Debug	Normal if PCSR $_n$ [DBG] cleared, stopped otherwise	No. Any $\overline{IRQ}_n$ Interrupt will be serviced upon normal exit from debug mode

In wait mode, the PIT module continues to operate as in run mode and can be configured to exit the low-power mode by generating an interrupt request. In doze mode with the PCSR $_n$ [DOZE] bit set, PIT module operation stops. In doze mode with the PCSR $_n$ [DOZE] bit cleared, doze mode does not affect PIT operation. When doze mode is exited, the PIT continues to operate in the state it was in prior to doze mode. In stop mode, the system clock is absent, and PIT module operation stops.

In debug mode with the PCSR $_n$ [DBG] bit set, PIT module operation stops. In debug mode with the PCSR $_n$ [DBG] bit cleared, debug mode does not affect PIT operation. When debug mode is exited, the PIT continues to operate in its pre-debug mode state, but any updates made in debug mode remain.

## 23.2 Memory Map/Register Definition

This section contains a memory map, shown in [Table 23-2](#), and describes the register structure for PIT0–PIT3.

**Table 23-2. Programmable Interrupt Timer Modules Memory Map**

IPSBAR Offset	[31:24]	[23:16]	[15:8]	[7:0]	Access <sup>1</sup>
0x15_0000	PIT Control and Status Register (PCSR0)		PIT Modulus Register (PMR0)		S
0x15_0004	PIT Count Register (PCNTR0)		Reserved <sup>2</sup>		S/U



**Table 23-2. Programmable Interrupt Timer Modules Memory Map (Continued)**

IPSBAR Offset	[31:24]	[23:16]	[15:8]	[7:0]	Access <sup>1</sup>
0x15_0008– 0x15_FFFF	Reserved				—
0x16_0000	PIT Control and Status Register (PCSR1)		PIT Modulus Register (PMR1)		S
0x16_0004	PIT Count Register (PCNTR1)		Reserved <sup>2</sup>		S/U
0x16_0008– 0x16_FFFF	Reserved				—
0x17_0000	PIT Control and Status Register (PCSR2)		PIT Modulus Register (PMR2)		S
0x17_0004	PIT Count Register (PCNTR2)		Reserved <sup>2</sup>		S/U
0x17_0008– 0x17_FFFF	Reserved				—
0x18_0000	PIT Control and Status Register (PCSR3)		PIT Modulus Register (PMR3)		S
0x18_0004	PIT Count Register (PCNTR3)		Reserved <sup>2</sup>		S/U
0x18_0008– 0x18_FFFF	Reserved				—

<sup>1</sup> S = CPU supervisor mode access only. S/U = CPU supervisor or user mode access. User mode accesses to supervisor only addresses have no effect and result in a cycle termination transfer error.

<sup>2</sup> Accesses to reserved address locations have no effect and result in a cycle termination transfer error.

## 23.2.1 Register Description

The PIT programming model consists of these registers:

- The PIT control and status register (PCSR<sub>*n*</sub>) configures the timer's operation.
- The PIT modulus register (PMR<sub>*n*</sub>) determines the timer modulus reload value.
- The PIT count register (PCNTR<sub>*n*</sub>) provides visibility to the counter value.

### 23.2.1.1 PIT Control and Status Register (PCSR<sub>*n*</sub>)

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	PRE				0	DOZE	DBG	OVW	PIE	PIF	RLD	EN
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Address	IPSBAR + 0x0015_0000 (PIT0); IPSBAR + 0x0016_0000 (PIT1); IPSBAR + 0x0017_0000 (PIT2); IPSBAR + 0x0018_0000 (PIT3)															

**Figure 23-2. PIT Control and Status Register (PCSR<sub>*n*</sub>)**

**Table 23-3. PCSR<sub>n</sub> Field Descriptions**

Bits	Name	Description																
15–12	—	Reserved, should be cleared.																
11–8	PRE	<p>Prescaler. The read/write prescaler bits select the system clock divisor to generate the PIT clock. To accurately predict the timing of the next count, change the PRE[3:0] bits only when the enable bit (EN) is clear. Changing PRE[3:0] resets the prescaler counter. System reset and the loading of a new value into the counter also reset the prescaler counter. Setting the EN bit and writing to PRE[3:0] can be done in this same write cycle. Clearing the EN bit stops the prescaler counter.</p> <table><tr><th>PRE</th><th>System Clock Divisor</th></tr><tr><td>0000</td><td>2<sup>0</sup></td></tr><tr><td>0001</td><td>2<sup>1</sup></td></tr><tr><td>0010</td><td>2<sup>2</sup></td></tr><tr><td>...</td><td>...</td></tr><tr><td>1101</td><td>2<sup>13</sup></td></tr><tr><td>1110</td><td>2<sup>14</sup></td></tr><tr><td>1111</td><td>2<sup>15</sup></td></tr></table>	PRE	System Clock Divisor	0000	2 <sup>0</sup>	0001	2 <sup>1</sup>	0010	2 <sup>2</sup>	...	...	1101	2 <sup>13</sup>	1110	2 <sup>14</sup>	1111	2 <sup>15</sup>
PRE	System Clock Divisor																	
0000	2 <sup>0</sup>																	
0001	2 <sup>1</sup>																	
0010	2 <sup>2</sup>																	
...	...																	
1101	2 <sup>13</sup>																	
1110	2 <sup>14</sup>																	
1111	2 <sup>15</sup>																	
7	—	Reserved, should be cleared.																
6	DOZE	<p>Doze mode bit. The read/write DOZE bit controls the function of the PIT in doze mode. Reset clears DOZE.</p> <p>0 PIT function not affected in doze mode 1 PIT function stopped in doze mode</p> <p>When doze mode is exited, timer operation continues from the state it was in before entering doze mode.</p>																
5	DBG	<p>Debug mode bit. Controls the function of the PIT in debug mode. Reset clears DBG. During debug mode, register read and write accesses function normally. When debug mode is exited, timer operation continues from the state it was in before entering debug mode, but any updates made in debug mode remain.</p> <p>0 PIT function not affected in debug mode 1 PIT function stopped in debug mode</p> <p>Note: Changing the DBG bit from 1 to 0 during debug mode starts the PIT timer. Likewise, changing the DBG bit from 0 to 1 during debug mode stops the PIT timer.</p>																
4	OVW	<p>Overwrite. Enables writing to PMR<sub>n</sub> to immediately overwrite the value in the PIT counter.</p> <p>0 Value in PMR<sub>n</sub> replaces value in PIT counter when count reaches 0x0000. 1 Writing PMR<sub>n</sub> immediately replaces value in PIT counter.</p>																
3	PIE	<p>PIT interrupt enable. This read/write bit enables the PIF flag to generate interrupt requests.</p> <p>0 PIF interrupt requests disabled 1 PIF interrupt requests enabled</p>																
2	PIF	<p>PIT interrupt flag. This read/write bit is set when the PIT counter reaches 0x0000. Clear PIF by writing a 1 to it or by writing to PMR. Writing 0 has no effect. Reset clears PIF.</p> <p>0 PIT count has not reached 0x0000. 1 PIT count has reached 0x0000.</p>																

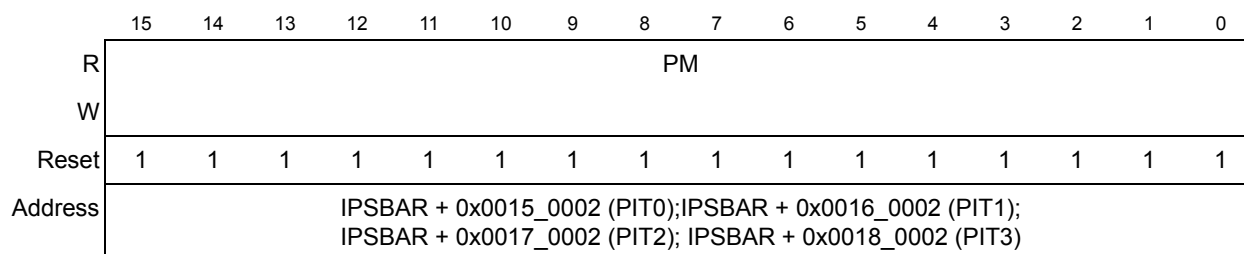
**Table 23-3. PCSR<sub>n</sub> Field Descriptions (Continued)**

Bits	Name	Description
1	RLD	Reload bit. The read/write reload bit enables loading the value of PMR <sub>n</sub> into the PIT counter when the count reaches 0x0000. 0 Counter rolls over to 0xFFFF on count of 0x0000 1 Counter reloaded from PMR <sub>n</sub> on count of 0x0000
0	EN	PIT enable bit. Enables PIT operation. When the PIT is disabled, the counter and prescaler are held in a stopped state. This bit is read anytime, write anytime. 0 PIT disabled 1 PIT enabled

### 23.2.1.2 PIT Modulus Register (PMR<sub>n</sub>)

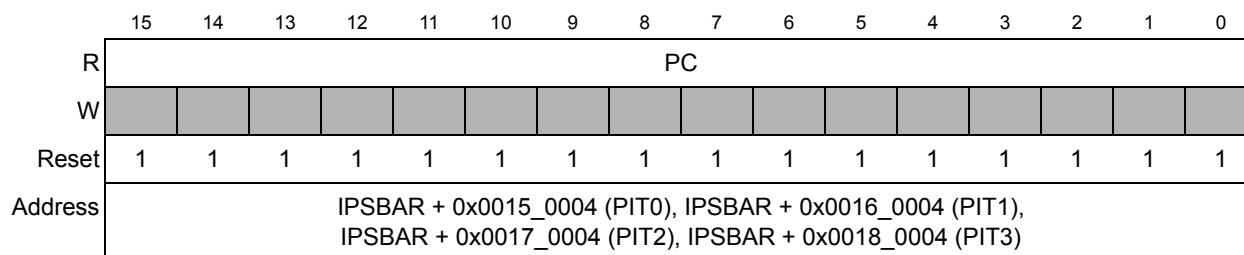
The 16-bit read/write PMR<sub>n</sub> contains the timer modulus value that is loaded into the PIT counter when the count reaches 0x0000 and the PCSR<sub>n</sub>[RLD] bit is set.

When the PCSR<sub>n</sub>[OVW] bit is set, PMR<sub>n</sub> is transparent, and the value written to PMR<sub>n</sub> is immediately loaded into the PIT counter. The prescaler counter is reset anytime a new value is loaded into the PIT counter and also during reset. Reading the PMR<sub>n</sub> returns the value written in the modulus latch. Reset initializes PMR<sub>n</sub> to 0xFFFF.

**Figure 23-3. PIT Modulus Register (PMR<sub>n</sub>)**

### 23.2.1.3 PIT Count Register (PCNTR<sub>n</sub>)

The 16-bit, read-only PCNTR<sub>n</sub> contains the counter value. Reading the 16-bit counter with two 8-bit reads is not guaranteed to be coherent. Writing to PCNTR<sub>n</sub> has no effect, and write cycles are terminated normally.

**Figure 23-4. PIT Count Register (PCNTR<sub>n</sub>)**

## 23.3 Functional Description

This section describes the PIT functional operation.

### 23.3.1 Set-and-Forget Timer Operation

This mode of operation is selected when the RLD bit in the PCSR register is set.

When the PIT counter reaches a count of 0x0000, the PIF flag is set in PCSR<sub>n</sub>. The value in the modulus register is loaded into the counter, and the counter begins decrementing toward 0x0000. If the PCSR<sub>n</sub>[PIE] bit is set, the PIF flag issues an interrupt request to the CPU.

When the PCSR<sub>n</sub>[OVW] bit is set, the counter can be directly initialized by writing to PMR<sub>n</sub> without having to wait for the count to reach 0x0000.

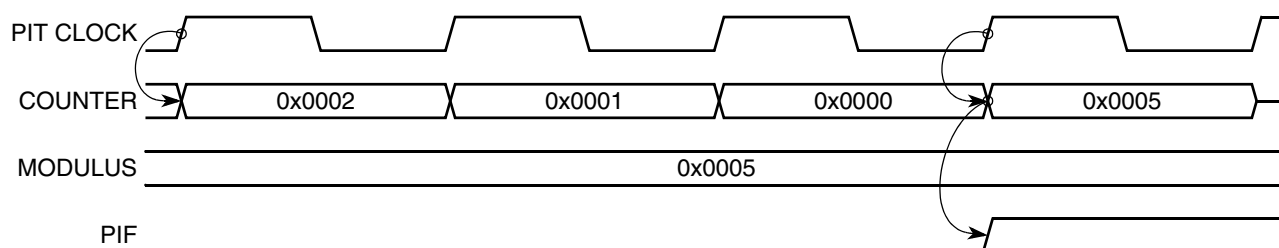


Figure 23-5. Counter Reloading from the Modulus Latch

### 23.3.2 Free-Running Timer Operation

This mode of operation is selected when the PCSR<sub>n</sub>[RLD] bit is clear. In this mode, the counter rolls over from 0x0000 to 0xFFFF without reloading from the modulus latch and continues to decrement.

When the counter reaches a count of 0x0000, the PCSR<sub>n</sub>[PIF] flag is set. If the PCSR<sub>n</sub>[PIE] bit is set, the PIF flag issues an interrupt request to the CPU.

When the PCSR<sub>n</sub>[OVW] bit is set, the counter can be directly initialized by writing to PMR<sub>n</sub> without having to wait for the count to reach 0x0000.

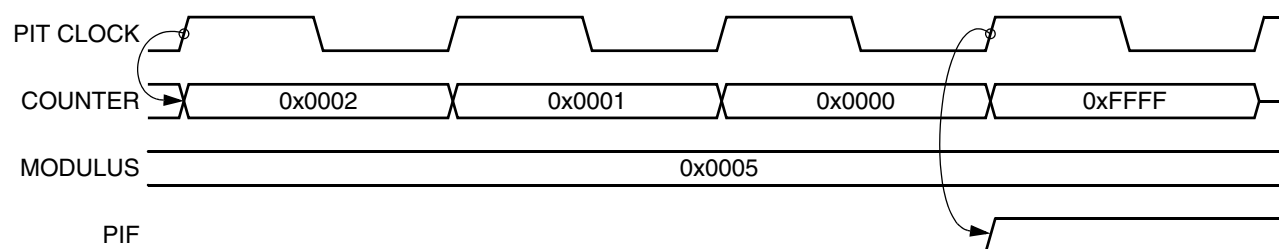


Figure 23-6. Counter in Free-Running Mode

### 23.3.3 Timeout Specifications

The 16-bit PIT counter and prescaler supports different timeout periods. The prescaler divides the internal bus clock period as selected by the PCSR $_n$ [PRE] bits. The PMR $_n$ [PM] bits select the timeout period.

$$\text{Timeout period} = \frac{\text{PRE}[3:0] \times (\text{PM}[15:0] + 1)}{f_{\text{sys}}/2}$$

### 23.3.4 Interrupt Operation

Table 23-4 shows the interrupt request generated by the PIT.

**Table 23-4. PIT Interrupt Requests**

Interrupt Request	Flag	Enable Bit
Timeout	PIF	PIE

The PIF flag is set when the PIT counter reaches 0x0000. The PIE bit enables the PIF flag to generate interrupt requests. Clear PIF by writing a 1 to it or by writing to the PMR.



# Chapter 24

## Queued Serial Peripheral Interface (QSPI) Module

### 24.1 Introduction

This chapter describes the queued serial peripheral interface (QSPI) module. Following a feature set overview is a description of operation including details of the QSPI's internal RAM organization. The chapter concludes with the programming model and a timing diagram.

#### 24.1.1 Overview

The queued serial peripheral interface module provides a serial peripheral interface with queued transfer capability. It allows users to queue up to 16 transfers at once, eliminating CPU intervention between transfers. Transfer RAM in the QSPI is indirectly accessible using address and data registers.

#### NOTE

The GPIO module must be configured to enable the peripheral function of the appropriate pins (refer to [Chapter 12, “General Purpose I/O Module”](#)) prior to configuring the QSPI Module.

#### 24.1.2 Features

- Programmable queue to support up to 16 transfers without user intervention
- Supports transfer sizes of 8 to 16 bits in 1-bit increments
- Four peripheral chip-select lines for control of up to 15 devices
- Baud rates from 162.7 Kbps to 20.75 Mbps at 83 MHz
- Programmable delays before and after transfers
- Programmable QSPI clock phase and polarity
- Supports wraparound mode for continuous transfers

#### 24.1.3 Module Description

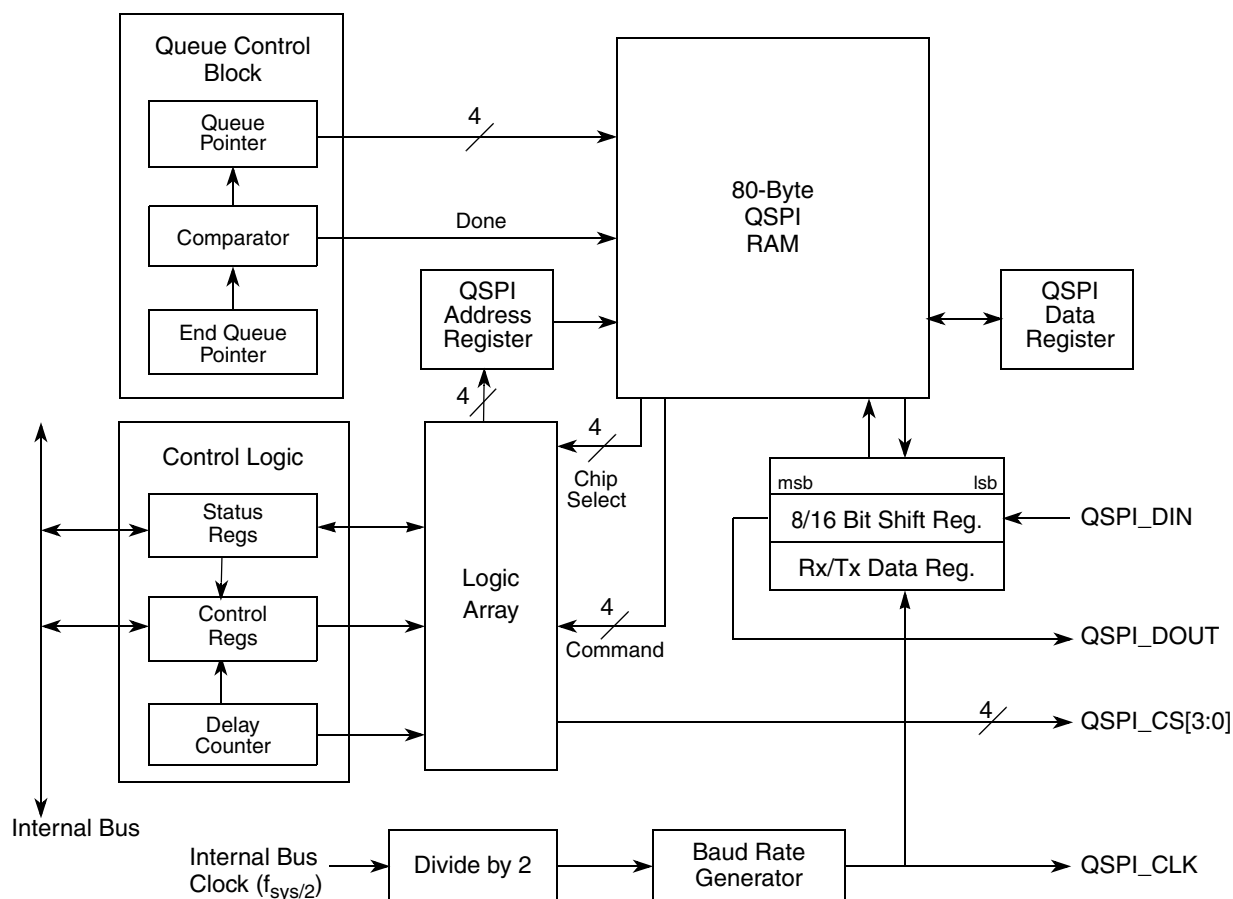
The QSPI module communicates with the integrated ColdFire CPU using internal memory mapped registers starting at IPSBAR + 0x340. See [Section 24.3, “Memory Map/Register Definition.”](#) A block diagram of the QSPI module is shown in [Figure 24-1](#).

### 24.1.3.1 Interface and Signals

The module provides access to as many as 15 devices with a total of seven signals: QSPI\_DOUT, QSPI\_DIN, QSPI\_CLK, QSPI\_CS0, QSPI\_CS1, QSPI\_CS2, and QSPI\_CS3.

Peripheral chip-select signals, QSPI\_CS[3:0], are used to select an external device as the source or destination for serial data transfer. Signals are asserted at a logic level corresponding to the value of the QSPI\_CS[3:0] bits in the command RAM whenever a command in the queue is executed. More than one chip-select signal can be asserted simultaneously.

Although QSPI\_CS[3:0] will function as simple chip selects in most applications, up to 15 devices can be selected by decoding them with an external 4-to-16 decoder.



**Figure 24-1. QSPI Block Diagram**

**Table 24-1. QSPI Input and Output Signals and Functions**

Signal Name	Hi-Z or Actively Driven	Function
QSPI Data Output (QSPI_DOUT)	Configurable	Serial data output from QSPI
QSPI Data Input (QSPI_DIN)	N/A	Serial data input to QSPI



**Table 24-1. QSPI Input and Output Signals and Functions (Continued)**

Signal Name	Hi-Z or Actively Driven	Function
Serial Clock (QSPI_CLK)	Actively driven	Clock output from QSPI
Peripheral Chip Selects (QSPI_CS[3:0])	Actively driven	Peripheral selects

### 24.1.4 Internal Bus Interface

Because the QSPI module only operates in master mode, the master bit in the QSPI mode register, QMR[MSTR], must be set for the QSPI to function properly. The QSPI can initiate serial transfers but cannot respond to transfers initiated by other QSPI masters.

## 24.2 Operation

The QSPI uses a dedicated 80-Byte block of static RAM accessible both to the module and the CPU to perform queued operations. The RAM is divided into three segments as follows:

- 16 command control bytes (command RAM)
- 16 transmit data words (transfer RAM)
- 16 receive data words (transfer RAM)

The RAM is organized so that 1 byte of command control data, 1 word of transmit data, and 1 word of receive data comprise 1 of the 16 queue entries (0x0–0xF).

### NOTE

Throughout ColdFire documentation, “word” is used consistently and exclusively to designate a 16-bit data unit. The only exceptions to this appear in discussions of serial communication modules such as QSPI that support variable-length data units. To simplify these discussions the functional unit is referred to as a ‘word’ regardless of length.

The user initiates QSPI operation by loading a queue of commands in command RAM, writing transmit data into transmit RAM, and then enabling the QSPI data transfer. The QSPI executes the queued commands and sets the completion flag in the QSPI interrupt register (QIR[SPIF]) to signal their completion. As another option, QIR[SPIFE] can be enabled to generate an interrupt.

The QSPI uses four queue pointers. The user can access three of them through fields in QSPI wrap register (QWR):

- The new queue pointer, QWR[NEWQP], points to the first command in the queue.
- An internal queue pointer points to the command currently being executed.
- The completed queue pointer, QWR[CPTQP], points to the last command executed.
- The end queue pointer, QWR[ENDQP], points to the final command in the queue.

The internal pointer is initialized to the same value as QWR[NEWQP]. During normal operation, the following sequence repeats:

1. The command pointed to by the internal pointer is executed.
2. The value in the internal pointer is copied into QWR[CPTQP].
3. The internal pointer is incremented.

Execution continues at the internal pointer address unless the QWR[NEWQP] value is changed. After each command is executed, QWR[ENDQP] and QWR[CPTQP] are compared. When a match occurs, QIR[SPIF] is set and the QSPI stops unless wraparound mode is enabled. Setting QWR[WREN] enables wraparound mode.

QWR[NEWQP] is cleared at reset. When the QSPI is enabled, execution begins at address 0x0 unless another value has been written into QWR[NEWQP]. QWR[ENDQP] is cleared at reset but is changed to show the last queue entry before the QSPI is enabled. QWR[NEWQP] and QWR[ENDQP] can be written at any time. When the QWR[NEWQP] value changes, the internal pointer value also changes unless a transfer is in progress, in which case the transfer completes normally. Leaving QWR[NEWQP] and QWR[ENDQP] set to 0x0 causes a single transfer to occur when the QSPI is enabled.

Data is transferred relative to QSPI\_CLK which can be generated in any one of four combinations of phase and polarity using QMR[CPHA,CPOL]. Data is transferred with the most significant bit (msb) first. The number of bits transferred defaults to 8, but can be set to any value between 8 and 16 by writing a value into the BITSE field of the command RAM (QCR[BITSE]).

## 24.2.1 QSPI RAM

The QSPI contains an 80-Byte block of static RAM that can be accessed by both the user and the QSPI. This RAM does not appear in the device memory map because it can only be accessed by the user indirectly through the QSPI address register (QAR) and the QSPI data register (QDR). The RAM is divided into three segments with 16 addresses each:

- Receive data RAM, the initial destination for all incoming data
- Transmit data RAM, a buffer for all out-bound data
- Command RAM, where commands are loaded

The transmit and command RAM are user write-only. The receive RAM is user read-only. [Figure 24-2](#) shows the RAM configuration. The RAM contents are undefined immediately after a reset.

The command and data RAM in the QSPI is indirectly accessible with QDR and QAR as 48 separate locations that comprise 16 words of transmit data, 16 words of receive data and 16 bytes of commands.

A write to QDR causes data to be written to the RAM entry specified by QAR[ADDR] and causes the value in QAR to increment. Correspondingly, a read at QDR returns the data in the RAM at the address specified by QAR[ADDR]. This also causes QAR to increment. A read access requires a single wait state.

Relative Address	Register	Function
0x00	QTR0	Transmit RAM  16 bits wide
0x01	QTR1	
...	...	
0x0F	QTR15	
0x10	QRR0	Receive RAM  16 bits wide
0x11	QRR1	
...	...	
0x1F	QRR15	
0x20	QCR0	Command RAM  8 bits wide
0x21	QCR1	
...	...	
0x2F	QCR15	

**Figure 24-2. QSPI RAM Model**

### 24.2.1.1 Receive RAM

Data received by the QSPI is stored in the receive RAM segment located at 0x10 to 0x1F in the QSPI RAM space. The user reads this segment to retrieve data from the QSPI. Data words with less than 16 bits are stored in the least significant bits of the RAM. Unused bits in a receive queue entry are set to zero upon completion of the individual queue entry.

QWR[CPTQP] shows which queue entries have been executed. The user can query this field to determine which locations in receive RAM contain valid data.

### 24.2.1.2 Transmit RAM

Data to be transmitted by the QSPI is stored in the transmit RAM segment located at addresses 0x0 to 0xF. The user normally writes 1 word into this segment for each queue command to be executed. The user cannot read data in the transmit RAM.

Outbound data must be written to transmit RAM in a right-justified format. The unused bits are ignored. The QSPI copies the data to its data serializer (shift register) for transmission. The data is transmitted most significant bit first and remains in transmit RAM until overwritten by the user.

### 24.2.1.3 Command RAM

The CPU writes one byte of control information to this segment for each QSPI command to be executed. Command RAM, referred to as QCR0–15, is write-only memory from a user's perspective.

Command RAM consists of 16 bytes with each byte divided into two fields. The peripheral chip select field controls the QSPI\_CS signal levels for the transfer. The command control field provides transfer options.

A maximum of 16 commands can be in the queue. Queue execution proceeds from the address in QWR[NEWQP] through the address in QWR[ENDQP].

The QSPI executes a queue of commands defined by the control bits in each command RAM entry which sequence the following actions:

- Chip-select pins are activated
- Data is transmitted from transmit RAM and received into the receive RAM
- The synchronous transfer clock QSPI\_CLK is generated

Before any data transfers begin, control data must be written to the command RAM, and any out-bound data must be written to transmit RAM. Also, the queue pointers must be initialized to the first and last entries in the command queue.

Data transfer is synchronized with the internally generated QSPI\_CLK, whose phase and polarity are controlled by QMR[CPHA] and QMR[CPOL]. These control bits determine which QSPI\_CLK edge is used to drive outgoing data and to latch incoming data.

### 24.2.2 Baud Rate Selection

The maximum QSPI clock frequency is one-fourth the clock frequency of the internal bus clock. Baud rate is selected by writing a value from 2–255 into QMR[BAUD]. The QSPI uses a prescaler to derive the QSPI\_CLK rate from the internal bus clock divided by two.

A baud rate value of zero turns off the QSPI\_CLK.

The desired QSPI\_CLK baud rate is related to the internal bus clock and QMR[BAUD] by the following expression:

$$\text{QMR[BAUD]} = f_{\text{sys}/2} / (2 \times [\text{desired QSPI\_CLK baud rate}])$$

**Table 24-2. QSPI\_CLK Frequency as Function of Internal Bus Clock and Baud Rate**

	Internal Bus Clock
QMR [BAUD]	83 MHz
2	20.75 MHz
4	10.375 MHz
8	5.188 MHz
16	2.594 MHz
32	1.297 MHz
255	129.4 kHz

### 24.2.3 Transfer Delays

The QSPI supports programmable delays for the QSPI\_CS signals before and after a transfer. The time between QSPI\_CS assertion and the leading QSPI\_CLK edge, and the time between the end of one transfer and the beginning of the next, are both independently programmable.

The chip select to clock delay enable bit in command RAM, QCR[DSCK], enables the programmable delay period from QSPI\_CS assertion until the leading edge of QSPI\_CLK. QDLYR[QCD] determines the period of delay before the leading edge of QSPI\_CLK. The following expression determines the actual delay before the QSPI\_CLK leading edge:

$$\text{QSPI\_CS-to-QSPI\_CLK delay} = \text{QDLYR[QCD]} / f_{\text{sys}/2}$$

QDLYR[QCD] has a range of 1–127.

When QDLYR[QCD] or QCR[DSCK] equals zero, the standard delay of one-half the QSPI\_CLK period is used.

The command RAM delay after transmit enable bit, QCR[DT], enables the programmable delay period from the negation of the QSPI\_CS signals until the start of the next transfer. The delay after transfer can be used to provide a peripheral deselect interval. A delay can also be inserted between consecutive transfers to allow serial A/D converters to complete conversion. There are two transfer delay options: the user can choose to delay a standard period after serial transfer is complete or can specify a delay period. Writing a value to QDLYR[DTL] specifies a delay period. QCR[DT] determines whether the standard delay period (DT = 0) or the specified delay period (DT = 1) is used. The following expression is used to calculate the delay:

$$\text{Delay after transfer} = 32 \times \text{QDLYR[DTL]} / f_{\text{sys}/2} \quad (\text{DT} = 1)$$

where QDLYR[DTL] has a range of 1–255.

A zero value for DTL causes a delay-after-transfer value of  $8192/f_{\text{sys}/2}$ .

$$\text{Standard delay after transfer} = 17/f_{\text{sys}/2} \quad (\text{DT} = 0)$$

Adequate delay between transfers must be specified for long data streams because the QSPI module requires time to load a transmit RAM entry for transfer. Receiving devices need at least the standard delay between successive transfers. If the internal bus clock is operating at a slower rate, the delay between transfers must be increased proportionately.

### 24.2.4 Transfer Length

There are two transfer length options. The user can choose a default value of 8 bits or a programmed value of 8 to 16 bits. The programmed value must be written into QMR[BITS]. The command RAM bits per transfer enable field, QCR[BITSE], determines whether the default value (BITSE = 0) or the BITS[3–0] value (BITSE = 1) is used. QMR[BITS] gives the required number of bits to be transferred, with 0b0000 representing 16.

### 24.2.5 Data Transfer

Operation is initiated by setting QDLYR[SPE]. Shortly after QDLYR[SPE] is set, the QSPI executes the command at the command RAM address pointed to by QWR[NEWQP]. Data at the pointer address in transmit RAM is loaded into the data serializer and transmitted. Data that is simultaneously received is stored at the pointer address in receive RAM.

When the proper number of bits has been transferred, the QSPI stores the working queue pointer value in QWR[CPTQP], increments the working queue pointer, and loads the next data for transfer from the transmit RAM. The command pointed to by the incremented working queue pointer is executed next unless a new value has been written to QWR[NEWQP]. If a new queue pointer value is written while a transfer is in progress, then that transfer is completed normally.

When the CONT bit in the command RAM is set, the QSPI\_CS signals are asserted between transfers. When CONT is cleared, QSPI\_CS[3:0] are negated between transfers. The QSPI\_CS signals are not high impedance.

When the QSPI reaches the end of the queue, it asserts the SPIF flag, QIR[SPIF]. If QIR[SPIFE] is set, an interrupt request is generated when QIR[SPIF] is asserted. Then the QSPI clears QDLYR[SPE] and stops, unless wraparound mode is enabled.

Wraparound mode is enabled by setting QWR[WREN]. The queue can wrap to pointer address 0x0, or to the address specified by QWR[NEWQP], depending on the state of QWR[WRT0].

In wraparound mode, the QSPI cycles through the queue continuously, even while requesting interrupt service. QDLYR[SPE] is not cleared when the last command in the queue is executed. New receive data overwrites previously received data in the receive RAM. Each time the end of the queue is reached, QIR[SPIFE] is set. QIR[SPIF] is not automatically reset. If interrupt driven QSPI service is used, the service routine must clear QIR[SPIF] to abort the current request. Additional interrupt requests during servicing can be prevented by clearing QIR[SPIFE].

There are two recommended methods of exiting wraparound mode: clearing QWR[WREN] or setting QWR[HALT]. Exiting wraparound mode by clearing QDLYR[SPE] is not recommended because this may abort a serial transfer in progress. The QSPI sets SPIF, clears QDLYR[SPE], and stops the first time it reaches the end of the queue after QWR[WREN] is cleared. After QWR[HALT] is set, the QSPI finishes the current transfer, then stops executing commands. After the QSPI stops, QDLYR[SPE] can be cleared.

## 24.3 Memory Map/Register Definition

Table 24-3 is the QSPI register memory map. Reading reserved locations returns zeros.

**Table 24-3. QSPI Registers**

IPSBAR Offset	[31:24]	[23:16]	[15:8]	[7:0]
0x00_0340	QSPI Mode Register (QMR)		Reserved <sup>1</sup>	
0x00_0344	QSPI Delay Register (QDLYR)		Reserved <sup>1</sup>	
0x00_0348	QSPI Wrap Register (QWR)		Reserved <sup>1</sup>	
0x00_034C	QSPI Interrupt Register (QIR)		Reserved <sup>1</sup>	
0x00_0350	QSPI Address Register (QAR)		Reserved <sup>1</sup>	
0x00_0354	QSPI Data Register (QDR)		Reserved <sup>1</sup>	

<sup>1</sup> Addresses not assigned to a register and undefined register bits are reserved for expansion. Write accesses to these reserved address spaces and reserved register bits have no effect.

### 24.3.1 QSPI Mode Register (QMR)

The QMR, shown in Figure 24-3, determines the basic operating modes of the QSPI module. Parameters such as QSPI\_CLK polarity and phase, baud rate, master mode operation, and transfer size are determined by this register. The data output high impedance enable, DOHIE, controls the operation of QSPI\_DOUT between data transfers. When DOHIE is cleared, QSPI\_DOUT is actively driven between transfers. When DOHIE is set, QSPI\_DOUT assumes a high impedance state.

#### NOTE

Because the QSPI does not operate in slave mode, the master mode enable bit, QMR[MSTR], must be set for the QSPI module to operate correctly.

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	MSTR	DOHIE	BITS				CPOL	CPHA	BAUD							
W																
Reset	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0
Address	IPSBAR + 0x00_0340															

Figure 24-3. QSPI Mode Register (QMR)

Table 24-4. QMR Field Descriptions

Bits	Name	Description																						
15	MSTR	Master mode enable. 0 Reserved, do not use. 1 The QSPI is in master mode. Must be set for the QSPI module to operate correctly.																						
14	DOHIE	Data output high impedance enable. Selects QSPI_DOUT mode of operation. 0 Default value after reset. QSPI_DOUT is actively driven between transfers. 1 QSPI_DOUT is high impedance between transfers.																						
13–10	BITS	Transfer size. Determines the number of bits to be transferred for each entry in the queue. <table><tr><th>BITS</th><th>Bits per Transfer</th></tr><tr><td>0000</td><td>16</td></tr><tr><td>0001–0111</td><td>Reserved</td></tr><tr><td>1000</td><td>8</td></tr><tr><td>1001</td><td>9</td></tr><tr><td>1010</td><td>10</td></tr><tr><td>1011</td><td>11</td></tr><tr><td>1100</td><td>12</td></tr><tr><td>1101</td><td>13</td></tr><tr><td>1110</td><td>14</td></tr><tr><td>1111</td><td>15</td></tr></table>	BITS	Bits per Transfer	0000	16	0001–0111	Reserved	1000	8	1001	9	1010	10	1011	11	1100	12	1101	13	1110	14	1111	15
BITS	Bits per Transfer																							
0000	16																							
0001–0111	Reserved																							
1000	8																							
1001	9																							
1010	10																							
1011	11																							
1100	12																							
1101	13																							
1110	14																							
1111	15																							
9	CPOL	Clock polarity. Defines the clock polarity of QSPI_CLK. 0 The inactive state value of QSPI_CLK is logic level 0. 1 The inactive state value of QSPI_CLK is logic level 1.																						
8	CPHA	Clock phase. Defines the QSPI_CLK clock-phase. 0 Data captured on the leading edge of QSPI_CLK and changed on the following edge of QSPI_CLK. 1 Data changed on the leading edge of QSPI_CLK and captured on the following edge of QSPI_CLK.																						
7–0	BAUD	Baud rate divider. The baud rate is selected by writing a value in the range 2–255. A value of zero disables the QSPI. A value of 1 is an invalid setting. The desired QSPI_CLK baud rate is related to the internal bus clock and QMR[BAUD] by the following expression: QMR[BAUD] = f <sub>sys</sub> /2 / (2 × [desired QSPI_CLK baud rate])																						



Figure 24-4 shows an example of a QSPI clocking and data transfer.

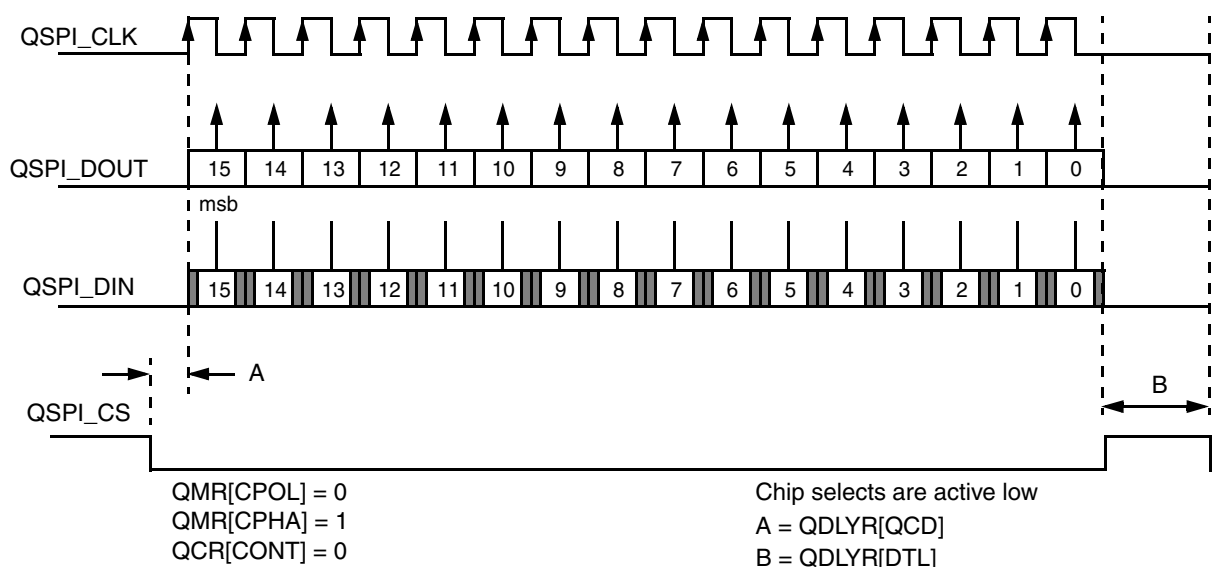


Figure 24-4. QSPI Clocking and Data Transfer Example

## 24.3.2 QSPI Delay Register (QDLYR)

Figure 24-5 shows the QDLYR.

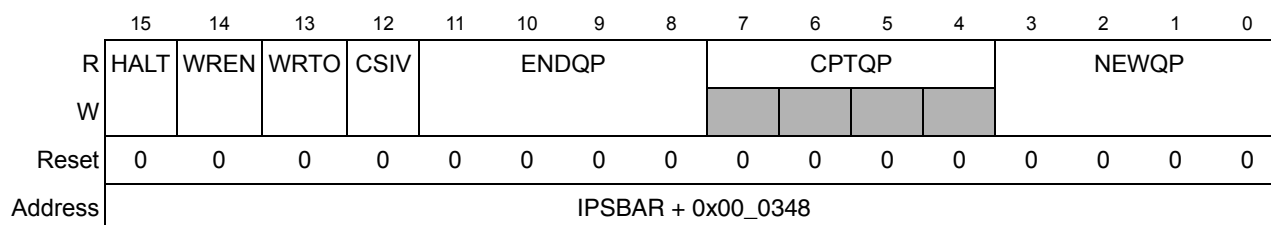
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	SPE	QCD							DTL							
W																
Reset	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0
Address	IPSBAR + 0x00_0344															

Figure 24-5. QSPI Delay Register (QDLYR)

Table 24-5. QDLYR Field Descriptions

Bits	Name	Description
15	SPE	QSPI enable. When set, the QSPI initiates transfers in master mode by executing commands in the command RAM. Automatically cleared by the QSPI when a transfer completes. The user can also clear this bit to abort transfer unless QIR[ABRTL] is set. The recommended method for aborting transfers is to set QWR[HALT].
14–8	QCD	QSPICLK delay. When the DSCK bit in the command RAM is set this field determines the length of the delay from assertion of the chip selects to valid QSPI_CLK transition.
7–0	DTL	Delay after transfer. When the DT bit in the command RAM is set this field determines the length of delay after the serial transfer.

## 24.3.3 QSPI Wrap Register (QWR)



**Figure 24-6. QSPI Wrap Register (QWR)**

**Table 24-6. QWR Field Descriptions**

Bits	Name	Description
15	HALT	Halt transfers. Assertion of this bit causes the QSPI to stop execution of commands once it has completed execution of the current command.
14	WREN	Wraparound enable. Enables wraparound mode. 0 Execution stops after executing the command pointed to by QWR[ENDQP]. 1 After executing command pointed to by QWR[ENDQP], wrap back to entry zero, or the entry pointed to by QWR[NEWQP] and continue execution.
13	WRTO	Wraparound location. Determines where the QSPI wraps to in wraparound mode. 0 Wrap to RAM entry zero. 1 Wrap to RAM entry pointed to by QWR[NEWQP].
12	CSIV	QSPI_CS inactive level. 0 QSPI chip select outputs return to zero when not driven from the value in the current command RAM entry during a transfer (that is, inactive state is 0, chip selects are active high). 1 QSPI chip select outputs return to one when not driven from the value in the current command RAM entry during a transfer (that is, inactive state is 1, chip selects are active low).
11–8	ENDQP	End of queue pointer. Points to the RAM entry that contains the last transfer description in the queue.
7–4	CPTQP	Completed queue entry pointer. Points to the RAM entry that contains the last command to have been completed. This field is read only.
3–0	NEWQP	Start of queue pointer. This 4-bit field points to the first entry in the RAM to be executed on initiating a transfer.

## 24.3.4 QSPI Interrupt Register (QIR)

Figure 24-7 shows the QIR.

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	WCEFB	ABRTB	0	ABRTL	WCEFE	ABRTE	0	SPIFE	0	0	0	0	WCEF	ABRT	0	SPIF
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Address	IPSBAR + 0x00_034C															

Figure 24-7. QSPI Interrupt Register (QIR)

Table 24-7. QIR Field Descriptions

Blts	Name	Description
15	WCEFB	Write collision access error enable. A write collision occurs during a data transfer when the RAM entry containing the command currently being executed is written to by the CPU with the QDR. When this bit is asserted, the write access to QDR results in an access error.
14	ABRTB	Abort access error enable. An abort occurs when QDLYR[SPE] is cleared during a transfer. When set, an attempt to clear QDLYR[SPE] during a transfer results in an access error.
13	—	Reserved, should be cleared.
12	ABRTL	Abort lock-out. When set, QDLYR[SPE] cannot be cleared by writing to the QDLYR. QDLYR[SPE] is only cleared by the QSPI when a transfer completes.
11	WCEFE	Write collision interrupt enable. Interrupt enable for WCEF. Setting this bit enables the interrupt, and clearing it disables the interrupt.
10	ABRTE	Abort interrupt enable. Interrupt enable for ABRT flag. Setting this bit enables the interrupt, and clearing it disables the interrupt.
9	—	Reserved, should be cleared.
8	SPIFE	QSPI finished interrupt enable. Interrupt enable for SPIF. Setting this bit enables the interrupt, and clearing it disables the interrupt.
7–4	—	Reserved, should be cleared.
3	WCEF	Write collision error flag. Indicates that an attempt has been made to write to the RAM entry that is currently being executed. Writing a 1 to this bit clears it and writing 0 has no effect.
2	ABRT	Abort flag. Indicates that QDLYR[SPE] has been cleared by the user writing to the QDLYR rather than by completion of the command queue by the QSPI. Writing a 1 to this bit clears it and writing 0 has no effect.
1	—	Reserved, should be cleared.
0	SPIF	QSPI finished flag. Asserted when the QSPI has completed all the commands in the queue. Set on completion of the command pointed to by QWR[ENDQP], and on completion of the current command after assertion of QWR[HALT]. In wraparound mode, this bit is set every time the command pointed to by QWR[ENDQP] is completed. Writing a 1 to this bit clears it and writing 0 has no effect.

The command and data RAM in the QSPI are indirectly accessible with QDR and QAR as 48 separate locations that comprise 16 words of transmit data, 16 words of receive data, and 16 bytes of commands.

A write to QDR causes data to be written to the RAM entry specified by QAR[ADDR]. This also causes the value in QAR to increment.

Correspondingly, a read at QDR returns the data in the RAM at the address specified by QAR[ADDR]. This also causes QAR to increment. A read access requires a single wait state.

### NOTE

The QAR does not wrap after the last queue entry within each section of the RAM. The application software must handle address range errors.

## 24.3.5 QSPI Address Register (QAR)

The QAR, shown in [Figure 24-8](#), is used to specify the location in the QSPI RAM that read and write operations affect.

	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	ADDR					
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Address	IPSBAR + 0x00_0350															

**Figure 24-8. QSPI Address Register**

## 24.3.6 QSPI Data Register (QDR)

The QDR, shown in [Figure 24-9](#), is used to access QSPI RAM indirectly. The CPU reads and writes all data from and to the QSPI RAM through this register.

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	DATA															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Address	IPSBAR + 0x00_0354															

**Figure 24-9. QSPI Data Register (QDR)**

## 24.3.7 Command RAM Registers (QCR0–QCR15)

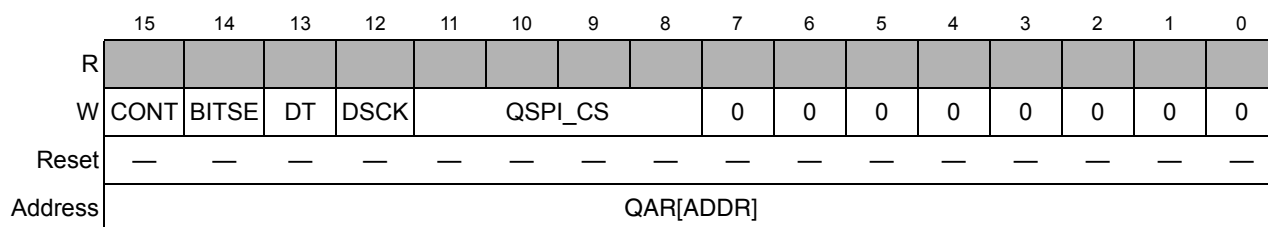
The command RAM is accessed using the upper byte of QDR. The QSPI cannot modify information in command RAM.

There are 16 bytes in the command RAM. Each byte is divided into two fields. The chip select field enables external peripherals for transfer. The command field provides transfer operations.

**NOTE**

The command RAM is accessed only using the most significant byte of QDR and indirect addressing based on QAR[ADDR].

Figure 24-10 shows the command RAM register.



**Figure 24-10. Command RAM Registers (QCR0–QCR15)**

**Table 24-8. QCR0–QCR15 Field Descriptions**

Bits	Name	Description
15	CONT	Continuous. 0 Chip selects return to inactive level defined by QWR[CSIV] when transfer is complete. 1 Chip selects remain asserted after the transfer of 16 words of data (see note below).
14	BITSE	Bits per transfer enable. 0 Eight bits 1 Number of bits set in QMR[BITS]
13	DT	Delay after transfer enable. 0 Default reset value. 1 The QSPI provides a variable delay at the end of serial transfer to facilitate interfacing with peripherals that have a latency requirement. The delay between transfers is determined by QDLYR[DTL].
12	DSCK	Chip select to QSPI_CLK delay enable. 0 Chip select valid to QSPI_CLK transition is one-half QSPI_CLK period. 1 QDLYR[QCD] specifies the delay from QSPI_CS valid to QSPI_CLK.
11–8	QSPI_CS	Peripheral chip selects. Used to select an external device for serial data transfer. More than one chip select may be active at once, and more than one device can be connected to each chip select. Bits 11–8 map directly to QSPI_CS[3:0], respectively. If it is desired to use those bits as a chip select value, then an external demultiplexor must be connected to the QSPI_CS[3:0] pins.
7–0	—	Reserved, should be cleared.

**NOTE**

In order to keep the chip selects asserted for all transfers, the QWR[CSIV] bit must be set to control the level that the chip selects return to after the first transfer.

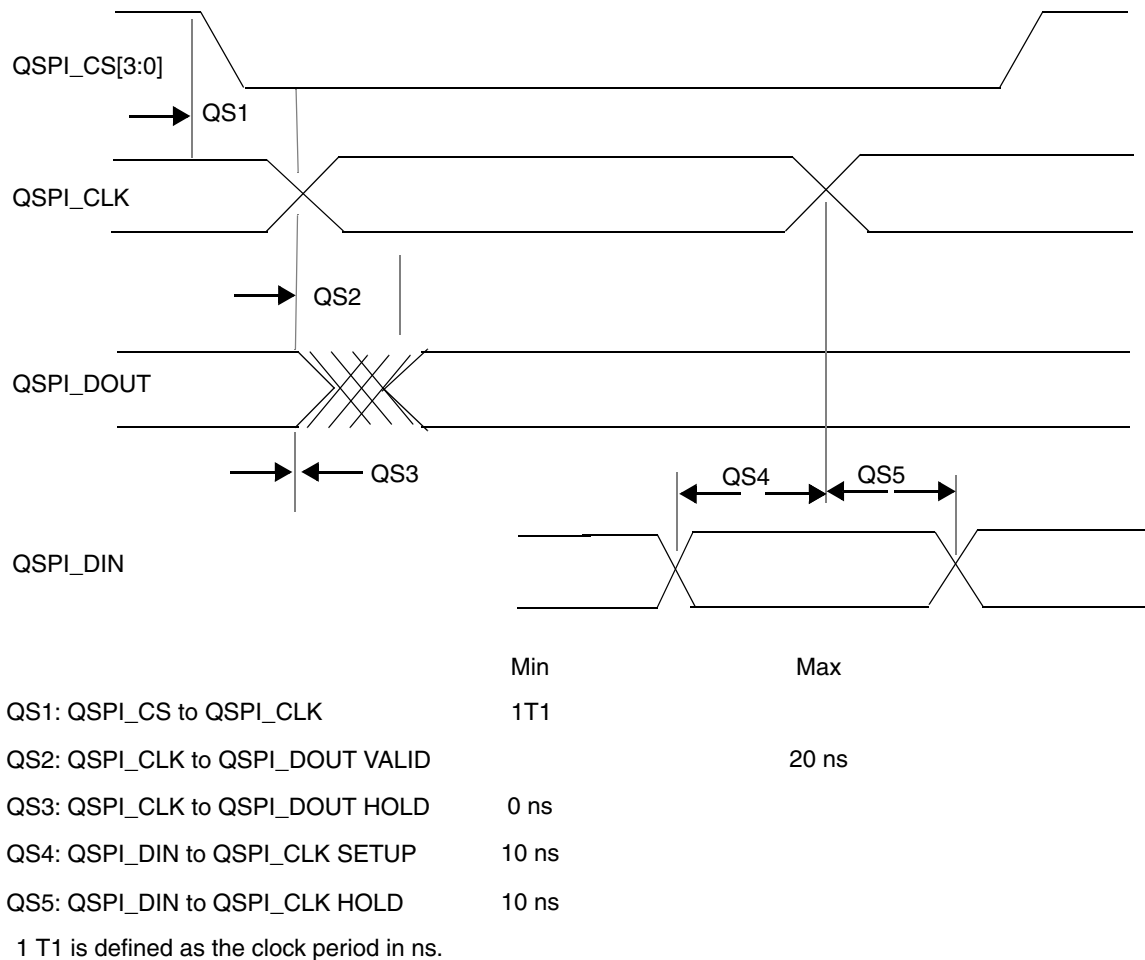


Figure 24-11. QSPI Timing

### 24.3.8 Programming Example

The following steps are necessary to set up the QSPI 12-bit data transfers and a QSPI\_CLK of 5.188 MHz. The QSPI RAM is set up for a queue of 16 transfers. All four QSPI\_CS signals are used in this example.

1. Write the QMR with 0xB308 to set up 12-bit data words with the data shifted on the falling clock edge, and a QSPI\_CLK frequency of 5.188 MHz (assuming a 83-MHz internal bus clock).
2. Write QDLYR with the desired delays.
3. Write QIR with 0xD00F to enable write collision, abort bus errors, and clear any interrupts.
4. Write QAR with 0x0020 to select the first command RAM entry.

5. Write QDR with 0x7E00, 0x7E00, 0x7E00, 0x7E00, 0x7D00, 0x7D00, 0x7D00, 0x7D00, 0x7B00, 0x7B00, 0x7B00, 0x7B00, 0x7700, 0x7700, 0x7700, and 0x7700 to set up four transfers for each chip select. The chip selects are active low in this example.
6. Write QAR with 0x0000 to select the first transmit RAM entry.
7. Write QDR with sixteen 12-bit words of data.
8. Write QWR with 0x0F00 to set up a queue beginning at entry 0 and ending at entry 15.
9. Set QDLR[SPE] to enable the transfers.
10. Wait until the transfers are complete. QIR[SPIF] is set when the transfers are complete.
11. Write QAR with 0x0010 to select the first receive RAM entry.
12. Read QDR to get the received data for each transfer.
13. Repeat steps 5 through 13 to do another transfer.





# Chapter 25

## DMA Timers (DTIM0–DTIM3)

### 25.1 Introduction

This chapter describes the configuration and operation of the four Direct Memory Access (DMA) timer modules (DTIM0, DTIM1, DTIM2, and DTIM3). These 32-bit timers provide input capture and reference compare capabilities with optional signaling of events using interrupts or triggers. Additionally, programming examples are included.

#### NOTE

The designation “*n*” is used throughout this section to refer to registers or signals associated with one of the four identical timer modules—DTIM0, DTIM1, DTIM2, or DTIM3.

#### 25.1.1 Overview

Each DMA timer module has a separate register set for configuration and control. The timers can be configured to operate from the system clock or from an external clocking source using the DTIN<sub>*n*</sub> signal. If the system clock is selected, it can be divided by 16 or 1. The selected clock source is routed to an 8-bit programmable prescaler that clocks the actual DMA timer counter register (DTCN<sub>*n*</sub>). Using the DTMR<sub>*n*</sub>, DTXMR<sub>*n*</sub>, DTCR<sub>*n*</sub>, and DTRR<sub>*n*</sub> registers, the DMA timer may be configured to assert an output signal, generate an interrupt, or initiate a DMA transfer on a particular event.

#### NOTE

The GPIO module must be configured to enable the peripheral function of the appropriate pins (refer to [Chapter 12, “General Purpose I/O Module”](#)) prior to configuring the DMA Timers.

[Figure 25-1](#) is a block diagram of one of the four identical timer modules.

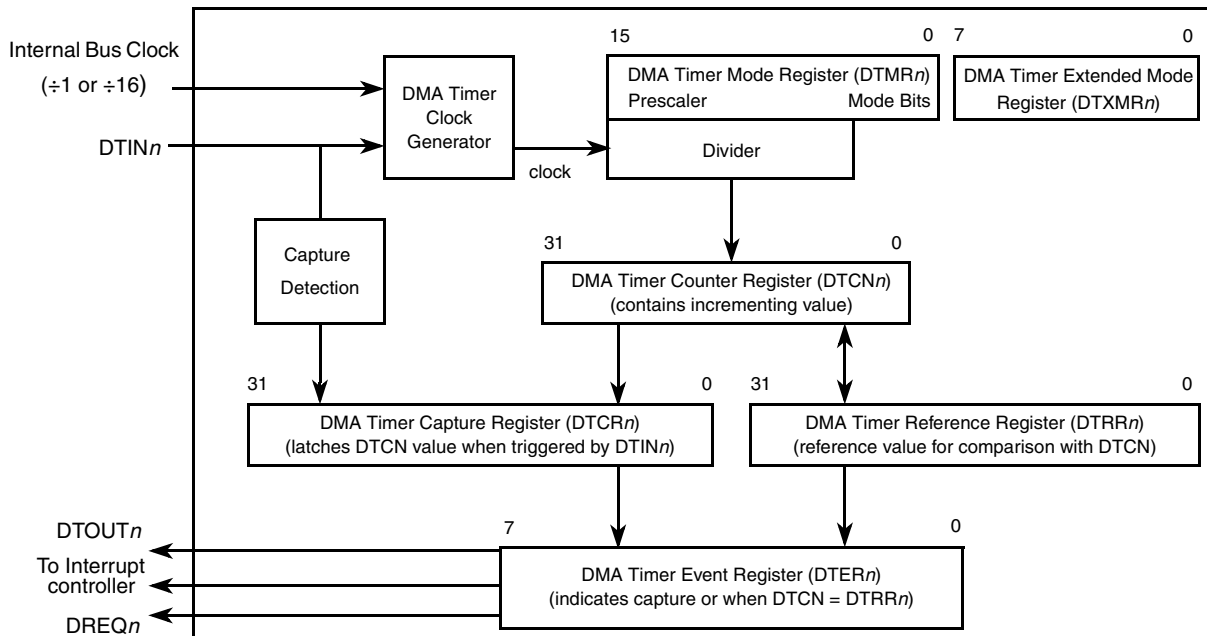


Figure 25-1. DMA Timer Block Diagram

## 25.1.2 Features

Each DMA timer module has the following features:

- Maximum timeout period of 211,954 seconds at 83 MHz (~659 hours)
- 12-ns resolution at 83 MHz
- Programmable sources for the clock input, including external clock
- Programmable prescaler
- Input-capture capability with programmable trigger edge on input pin
- Programmable mode for the output pin on reference compare
- Free run and restart modes
- Programmable interrupt or DMA request on input capture or reference-compare

## 25.2 Memory Map/Register Definition

The following features are programmable through the timer registers, shown in [Table 25-1](#):

### 25.2.1 Prescaler

The prescaler clock input is selected from system clock (divided by 1 or 16) or from the corresponding timer input, DTIN<sub>n</sub>. DTIN<sub>n</sub> is synchronized to the system clock. The

synchronization delay is between two and three system clocks. The corresponding DTMR $n$ [CLK] selects the clock input source. A programmable prescaler divides the clock input by values from 1 to 256. The prescaler output is an input to the 32-bit counter, DTCN $n$ .

## 25.2.2 Capture Mode

Each DMA timer has a 32-bit timer capture register (DTCR $n$ ) that latches the counter value when the corresponding input capture edge detector senses a defined DTIN $n$  transition. The capture edge bits (DTMR $n$ [CE]) select the type of transition that triggers the capture and sets the timer event register capture event bit, DTER $n$ [CAP]. If DTER $n$ [CAP] is set and DTXMR $n$ [DMAEN] is one, a DMA request is asserted. If DTER $n$ [CAP] is set and DTXMR $n$ [DMAEN] is zero, an interrupt is asserted.

## 25.2.3 Reference Compare

Each DMA timer can be configured to count up to a reference value, at which point DTER $n$ [REF] is set. If DTMR $n$ [ORRI] is one and DTXMR $n$ [DMAEN] is zero, an interrupt is asserted. If DTMR $n$ [ORRI] is one and DTXMR $n$ [DMAEN] is one, a DMA request is asserted. If the free run/restart bit DTMR $n$ [FRR] is set, a new count starts. If it is clear, the timer keeps running.

## 25.2.4 Output Mode

When a timer reaches the reference value selected by DTRR, it can send an output signal on DTOUT $n$ . DTOUT $n$  can be an active-low pulse or a toggle of the current output as selected by the DTMR $n$ [OM] bit.

## 25.2.5 Memory Map

The timer module registers, shown in [Table 25-1](#), can be modified at any time.

**Table 25-1. DMA Timer Module Memory Map**

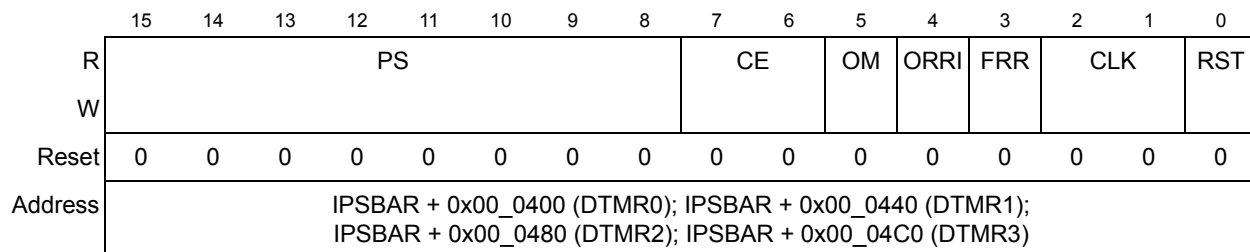
IPSBAR Offset	[31:24]	[23:16]	[15:8]	[7:0]
0x00_0400	DMA Timer0 Mode Register (DTMR0)		DMA Timer0 Extended Mode Register (DTXMR0)	DMA Timer0 Event Register (DTER0)
0x00_0404	DMA Timer0 Reference Register (DTRR0)			
0x00_0408	DMA Timer0 Capture Register (DTCR0)			
0x00_040C	DMA Timer0 Counter Register (DTCN0)			
0x00_0440	DMA Timer1 Mode Register (DTMR1)		DMA Timer1 Extended Mode Register (DTXMR1)	DMA Timer1 Event Register (DTER1)

**Table 25-1. DMA Timer Module Memory Map (Continued)**

IPSBAR Offset	[31:24]	[23:16]	[15:8]	[7:0]
0x00_0444	DMA Timer1 Reference Register (DTRR1)			
0x00_0448	DMA Timer1 Capture Register (DTCR1)			
0x00_044C	DMA Timer1 Counter Register (DTCN1)			
0x00_0480	DMA Timer2 Mode Register (DTMR2)		DMA Timer2 Extended Mode Register (DTXMR2)	DMA Timer2 Event Register (DTER2)
0x00_0484	DMA Timer2 Reference Register (DTRR2)			
0x00_0488	DMA Timer2 Capture Register (DTCR2)			
0x00_048C	DMA Timer2 Counter Register (DTCN2)			
0x00_04C0	DMA Timer3 Mode Register (DTMR3)		DMA Timer3 Extended Mode Register (DTXMR3)	DMA Timer3 Event Register (DTER3)
0x00_04C4	DMA Timer3 Reference Register (DTRR3)			
0x00_04C8	DMA Timer3 Capture Register (DTCR3)			
0x00_04CC	DMA Timer3 Counter Register (DTCN3)			

## 25.2.6 DMA Timer Mode Registers (DTMR<sub>n</sub>)

DTMRs, shown in [Figure 25-2](#), program the prescaler and various timer modes.

**Figure 25-2. DMA Timer Mode Registers (DTMR<sub>n</sub>)****Table 25-2. DTMR<sub>n</sub> Field Descriptions**

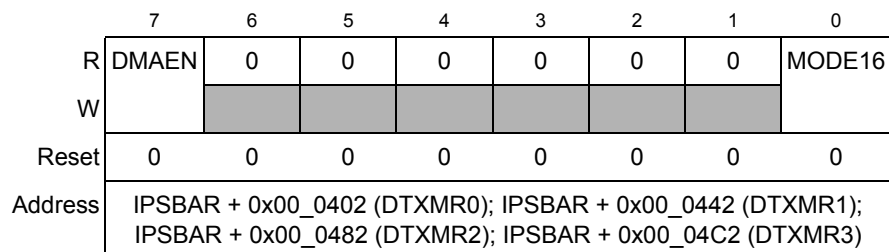
Bits	Name	Description
15–8	PS	Prescaler value. The prescaler is programmed to divide the clock input (system clock/(16 or 1) or clock on DTIN <sub>n</sub> ) by values from 1 (PS = 0x00) to 256 (PS = 0xFF).
7–6	CE	Capture edge. 00 Disable capture event output 01 Capture on rising edge only 10 Capture on falling edge only 11 Capture on any edge

**Table 25-2. DTMR<sub>n</sub> Field Descriptions (Continued)**

Bits	Name	Description
5	OM	Output mode. 0 Active-low pulse for one system clock cycle (12-ns resolution at 83 MHz). 1 Toggle output.
4	ORRI	Output reference request, interrupt enable. If ORRI is set when DTER <sub>n</sub> [REF] = 1, a DMA request or an interrupt occurs, depending on the value of DTXMR <sub>n</sub> [DMAEN] (DMA request if =1, interrupt if =0). 0 Disable DMA request or interrupt for reference reached (does not affect DMA request or interrupt on capture function). 1 Enable DMA request or interrupt upon reaching the reference value.
3	FRR	Free run/restart 0 Free run. Timer count continues to increment after reaching the reference value. 1 Restart. Timer count is reset immediately after reaching the reference value.
2–1	CLK	Input clock source for the timer 00 Stop count 01 System clock divided by 1 10 System clock divided by 16. Note that this clock source is not synchronized with the timer; thus successive time-outs may vary slightly. 11 DTIN <sub>n</sub> pin (falling edge)
0	RST	Reset timer. Performs a software timer reset similar to an external reset, although other register values can still be written while RST = 0. A transition of RST from 1 to 0 resets register values. The timer counter is not clocked unless the timer is enabled. 0 Reset timer (software reset) 1 Enable timer

## 25.2.7 DMA Timer Extended Mode Registers (DTXMR<sub>n</sub>)

DTXMR<sub>n</sub>, shown in [Figure 25-3](#), program DMA request and increment modes for the timers.

**Figure 25-3. DMA Timer Extended Mode Registers (DTXMR<sub>n</sub>)****Table 25-3. DTXMR<sub>n</sub> Field Descriptions**

Bits	Name	Description
7	DMAEN	DMA request. Enables DMA request output on counter reference match or capture edge event. 0 DMA request disabled 1 DMA request enabled

**Table 25-3. DTXMR<sub>n</sub> Field Descriptions (Continued)**

Bits	Name	Description
6–1	—	Reserved, should be cleared.
0	MODE16	<p>Selects the increment mode for the timer. MODE16 = 1 is intended to exercise the upper bits of the 32-bit timer in diagnostic software without requiring the timer to count through its entire dynamic range. When set, the counter's upper 16 bits mirror its lower 16 bits. All 32 bits of the counter are still compared to the reference value.</p> <p>0 Increment timer by 1 1 Increment timer by 65,537</p>

### 25.2.8 DMA Timer Event Registers (DTER<sub>n</sub>)

DTER<sub>n</sub>, shown in [Figure 25-4](#), reports capture or reference events by setting DTER<sub>n</sub>[CAP] or DTER<sub>n</sub>[REF]. This reporting is done regardless of the corresponding DMA request or interrupt enable values, DTXMR<sub>n</sub>[DMAEN] and DTMR<sub>n</sub>[ORRI,CE].

Writing a 1 to either DTER<sub>n</sub>[REF] or DTER<sub>n</sub>[CAP] clears it (writing a 0 does not affect bit value); both bits can be cleared at the same time. If configured to generate an interrupt request, the REF and CAP bits must be cleared early in the interrupt service routine so the timer module can negate the interrupt request signal to the interrupt controller. If configured to generate a DMA request, the processing of the DMA data transfer automatically clears both the REF and CAP flags via the internal DMA ACK signal.

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	REF	CAP
W							w1c	w1c
Reset	0	0	0	0	0	0	0	0
Address	IPSBAR + 0x00_0403 (DTER0); IPSBAR + 0x00_0443 (DTER1); IPSBAR + 0x00_0483 (DTER2); IPSBAR + 0x00_04C3 (DTER3)							

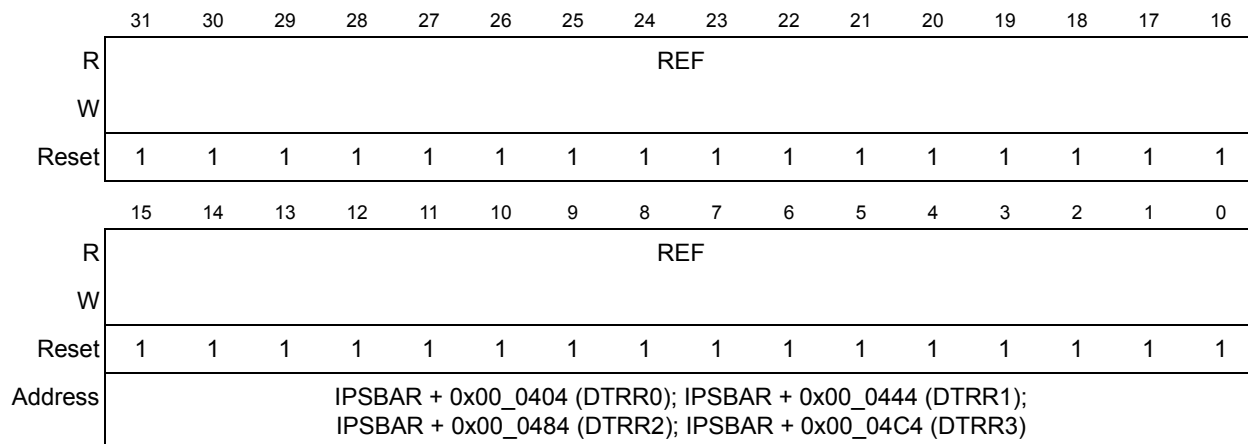
**Figure 25-4. DMA Timer Event Registers (DTER<sub>n</sub>)**

**Table 25-4. DTER<sub>n</sub> Field Descriptions**

Bits	Name	Description																																								
7–2	—	Reserved, should be cleared.																																								
1	REF	<p>Output reference event. The counter value, DTCN<sub>n</sub>, equals the reference value, DTRR<sub>n</sub>. Writing a one to REF clears the event condition. Writing a zero has no effect.</p> <table><tr><th>REF</th><th>DTMR<sub>n</sub>[ORRI]</th><th>DTXMR<sub>n</sub>[DMAEN]</th><th></th></tr><tr><td>0</td><td>X</td><td>X</td><td>No event</td></tr><tr><td>1</td><td>0</td><td>0</td><td>No request asserted</td></tr><tr><td>1</td><td>0</td><td>1</td><td>No request asserted</td></tr><tr><td>1</td><td>1</td><td>0</td><td>Interrupt request asserted</td></tr><tr><td>1</td><td>1</td><td>1</td><td>DMA request asserted</td></tr></table>	REF	DTMR <sub>n</sub> [ORRI]	DTXMR <sub>n</sub> [DMAEN]		0	X	X	No event	1	0	0	No request asserted	1	0	1	No request asserted	1	1	0	Interrupt request asserted	1	1	1	DMA request asserted																
REF	DTMR <sub>n</sub> [ORRI]	DTXMR <sub>n</sub> [DMAEN]																																								
0	X	X	No event																																							
1	0	0	No request asserted																																							
1	0	1	No request asserted																																							
1	1	0	Interrupt request asserted																																							
1	1	1	DMA request asserted																																							
0	CAP	<p>Capture event. The counter value has been latched into DTCR<sub>n</sub>. Writing a one to CAP clears the event condition. Writing a zero has no effect.</p> <table><tr><th>CAP</th><th>DTMR<sub>n</sub>[CE]</th><th>DTXMR<sub>n</sub>[DMAEN]</th><th></th></tr><tr><td>0</td><td>X</td><td>X</td><td>No event</td></tr><tr><td>1</td><td>00</td><td>0</td><td>Disable capture event output</td></tr><tr><td>1</td><td>00</td><td>1</td><td>Disable capture event output</td></tr><tr><td>1</td><td>01</td><td>0</td><td>Capture on rising edge &amp; trigger interrupt</td></tr><tr><td>1</td><td>01</td><td>1</td><td>Capture on rising edge &amp; trigger DMA</td></tr><tr><td>1</td><td>10</td><td>0</td><td>Capture on falling edge &amp; trigger interrupt</td></tr><tr><td>1</td><td>10</td><td>1</td><td>Capture on falling edge &amp; trigger DMA</td></tr><tr><td>1</td><td>11</td><td>0</td><td>Capture on any edge &amp; trigger interrupt</td></tr><tr><td>1</td><td>11</td><td>1</td><td>Capture on any edge &amp; trigger DMA</td></tr></table>	CAP	DTMR <sub>n</sub> [CE]	DTXMR <sub>n</sub> [DMAEN]		0	X	X	No event	1	00	0	Disable capture event output	1	00	1	Disable capture event output	1	01	0	Capture on rising edge & trigger interrupt	1	01	1	Capture on rising edge & trigger DMA	1	10	0	Capture on falling edge & trigger interrupt	1	10	1	Capture on falling edge & trigger DMA	1	11	0	Capture on any edge & trigger interrupt	1	11	1	Capture on any edge & trigger DMA
CAP	DTMR <sub>n</sub> [CE]	DTXMR <sub>n</sub> [DMAEN]																																								
0	X	X	No event																																							
1	00	0	Disable capture event output																																							
1	00	1	Disable capture event output																																							
1	01	0	Capture on rising edge & trigger interrupt																																							
1	01	1	Capture on rising edge & trigger DMA																																							
1	10	0	Capture on falling edge & trigger interrupt																																							
1	10	1	Capture on falling edge & trigger DMA																																							
1	11	0	Capture on any edge & trigger interrupt																																							
1	11	1	Capture on any edge & trigger DMA																																							

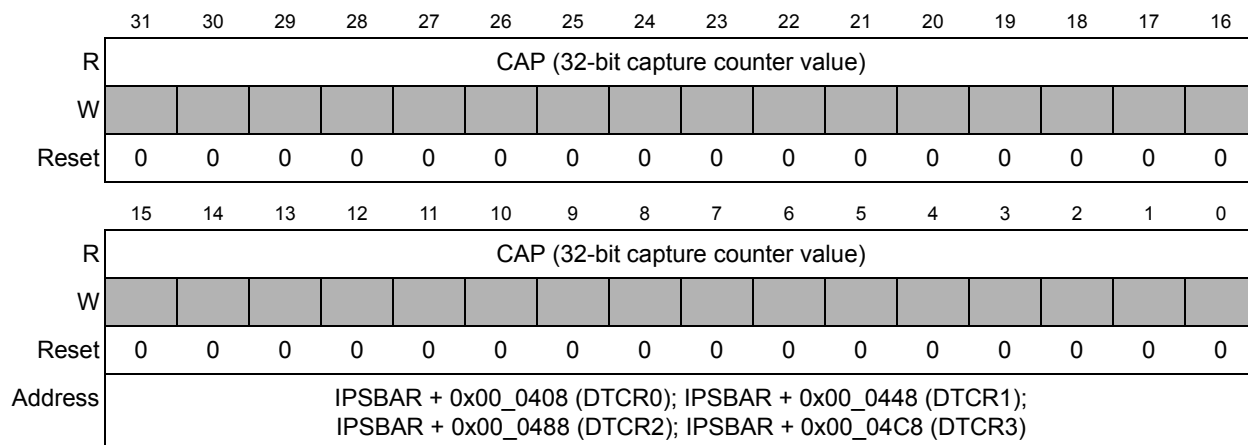
### 25.2.9 DMA Timer Reference Registers (DTRR<sub>n</sub>)

Each DTRR<sub>n</sub>, shown in [Figure 25-5](#), contains the reference value compared with the respective free-running timer counter (DTCN<sub>n</sub>) as part of the output-compare function. The reference value is not matched until DTCN<sub>n</sub> equals DTRR<sub>n</sub>, and the prescaler indicates that DTCN<sub>n</sub> should be incremented again. Thus, the reference register is matched after DTRR<sub>n</sub>+1 time intervals.

Figure 25-5. DMA Timer Reference Registers (DTRR $n$ )

### 25.2.10 DMA Timer Capture Registers (DTCR $n$ )

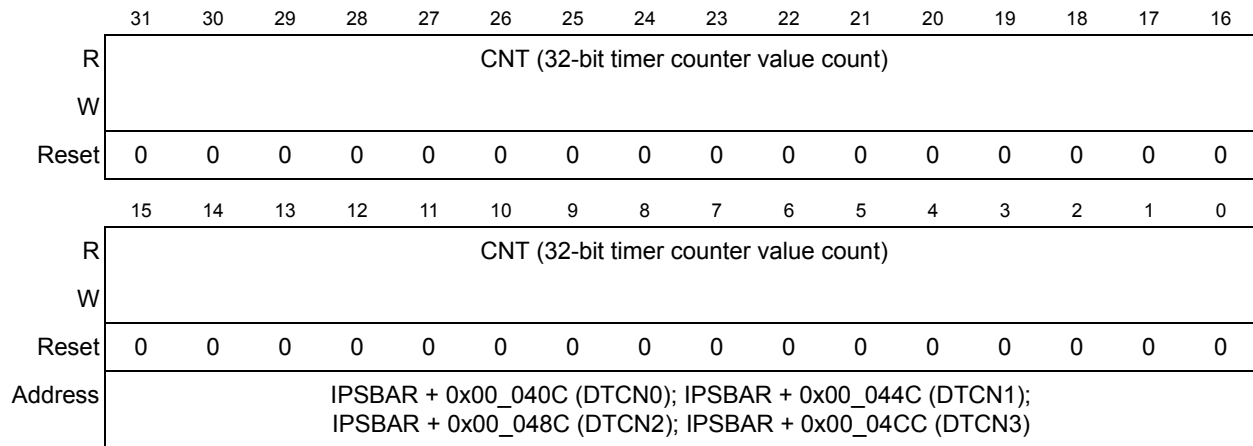
Each DTCR $n$ , shown in Figure 25-6, latches the corresponding DTCN $n$  value during a capture operation when an edge occurs on DTIN $n$ , as programmed in DTMR $n$ . The system clock is assumed to be the clock source. DTIN $n$  cannot simultaneously function as a clocking source and as an input capture pin. Indeterminate operation will result if DTIN $n$  is set as the clock source when the input capture mode is used.

Figure 25-6. DMA Timer Capture Registers (DTCR $n$ )

### 25.2.11 DMA Timer Counters (DTCN $n$ )

The current value of the 32-bit DTCNs can be read at anytime without affecting counting. Writing to DTCN $n$ , shown in Figure 25-7, clears it. The timer counter increments on the clock source rising edge (system clock  $\div$  1, system clock  $\div$  16, or DTIN $n$ ).



Figure 25-7. DMA Timer Counters (DTCN $n$ )

## 25.3 Using the DMA Timer Modules

The general-purpose timer modules are typically used in the following manner, though this is not necessarily the program order in which these actions must occur:

- The DTMR $n$  and DTXMR $n$  registers are configured for the desired function and behavior.
  - Count and compare to a reference value stored in the DTRR $n$  register
  - Capture the timer value on an edge detected on DTIN $n$
  - Configure DTOUT $n$  output mode
  - Increment counter by 1 or by 65,537 (16-bit mode)
  - Enable/disable interrupt or DMA request on counter reference match or capture edge
- The DTMR $n$ [CLK] register is configured to select the clock source to be routed to the prescaler.
  - System clock (can be divided by 1 or 16)
  - DTIN $n$ , the maximum value of DTIN $n$  is 1/5 of the system clock, as described in the MCF5275 Electrical Characteristics.

### NOTE

DTIN $n$  may not be configured as a clock source when the timer capture mode is selected or indeterminate operation will result.

- The 8-bit DTMR $n$ [PS] prescaler value is set
- Using DTMR $n$ [RST] the counter is cleared and started
- Timer events are either handled with an interrupt service routine, a DMA request, or by a software polling mechanism

## 25.3.1 Code Example

The following code provides an example of how to initialize DMA Timer0 and how to use the timer for counting time-out periods.

```
DTMR0 EQU IPSBARx+0x400;Timer0 mode register
DTMR1 EQU IPSBARx+0x440 ;Timer1 mode register
DTRR0 EQU IPSBARx+0x404 ;Timer0 reference register
DTRR1 EQU IPSBARx+0x444 ;Timer1 reference register
DTCR0 EQU IPSBARx+0x408 ;Timer0 capture register
DTCR1 EQU IPSBARx+0x448 ;Timer1 capture register
DTCN0 EQU IPSBARx+0x40C ;Timer0 counter register
DTCN1 EQU IPSBARx+0x44C ;Timer1 counter register
DTER0 EQU IPSBARx+0x403 ;Timer0 event register
DTER1 EQU IPSBARx+0x443 ;Timer1 event register

* TMR0 is defined as: *
*[PS] = 0xFF,    divide clock by 256
*[CE] = 00       disable capture event output
*[OM] = 0        output=active-low pulse
*[ORRI] = 0,     disable ref. match output
*[FRR] = 1,      restart mode enabled
*[CLK] = 10,     system clock/16
*[RST] = 0,      timer0 disabled

        move.w #0xFF0C,D0
        move.w D0,TMR0

        move.l #0x0000,D0;writing to the timer counter with any
        move.l D0,TCN0 ;value resets it to zero

        move.l #AFAF,D0 ;set the timer0 reference to be
        move.l #D0,TRR0 ;defined as 0xAFAF
```

The simple example below uses Timer0 to count time-out loops. A time-out occurs when the reference value, 0xAFAF, is reached.

```
timer0_ex
        clr.l D0
        clr.l D1
        clr.l D2

        move.l #0x0000,D0
        move.l D0,TCN0           ;reset the counter to 0x0000

        move.b #0x03,D0          ;writing ones to TER0[REF,CAP]
        move.b D0,TER0           ;clears the event flags

        move.w TMR0,D0           ;save the contents of TMR0 while setting
        bset #0,D0               ;the 0 bit. This enables timer 0 and starts counting
        move.w D0,TMR0           ;load the value back into the register, setting
TMR0[RST]

T0_LOOP
        move.b TER0,D1           ;load TER0 and see if
```

```

        btst #1,D1                ;TER0[REF] has been set
        beq T0_LOOP

        addi.l #1,D2              ;Increment D2
        cmp.l #5,D2              ;Did D2 reach 5? (i.e. timer ref has timed)
        beq T0_FINISH            ;If so, end timer0 example. Otherwise jump
back.

        move.b #0x02,D0           ;writing one to TER0[REF] clears the event
flag

        move.b D0,TER0
        jmp T0_LOOP

T0_FINISH
        HALT                     ;End processing. Example is finished

```

## 25.3.2 Calculating Time-Out Values

The formula below determines time-out periods for various reference values:

$$\text{Time-out period} = (1/\text{clock frequency}) \times (1 \text{ or } 16) \times (\text{DTMR}_n[\text{PS}] + 1) \times (\text{DTRR}_n[\text{REF}] + 1)$$

When calculating time-out periods, add 1 to the prescaler to simplify calculating, because  $\text{DTMR}_n[\text{PS}] = 0x00$  yields a prescaler of 1 and  $\text{DTMR}_n[\text{PS}] = 0xFF$  yields a prescaler of 256.

For example, if a 83-MHz timer clock is divided by 16,  $\text{DTMR}_n[\text{PS}] = 0x7F$ , and the timer is referenced at  $0x13C9E$  (81,054 decimal), the time-out period is as follows:

$$\text{Time-out period} = [1/(83 \times 10^6)] \times (16) \times (127 + 1) \times (81,054 + 1) = 2.00 \text{ s}$$



# Chapter 26

## I<sup>2</sup>C Interface

### 26.1 Introduction

This chapter describes the MCF5275 I<sup>2</sup>C module, including I<sup>2</sup>C protocol, clock synchronization, and I<sup>2</sup>C programming model registers. It also provides extensive programming examples.

### 26.2 Overview

I<sup>2</sup>C is a two-wire, bidirectional serial bus that provides a simple, efficient method of data exchange, minimizing the interconnection between devices. This bus is suitable for applications that require occasional communication between many devices over a short distance. The flexible I<sup>2</sup>C bus allows additional devices to be connected to the bus for expansion and system development.

The interface is designed to operate up to 100 Kbps with maximum bus loading and timing. The device is capable of operating at higher baud rates, up to a maximum of the internal bus clock divided by 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 I<sup>2</sup>C system is a true multiple-master bus; it uses arbitration and collision detection to prevent data corruption in the event that multiple devices attempt to control the bus simultaneously. This feature supports complex applications with multiprocessor control and can be used for rapid testing and alignment of end products through external connections to an assembly-line computer.

#### NOTE

The I<sup>2</sup>C module is designed to be compatible with the Philips I<sup>2</sup>C bus protocol. For information on system configuration, protocol, and restrictions, see *The I<sup>2</sup>C Bus Specification, Version 2.1*.

#### NOTE

The GPIO module must be configured to enable the peripheral function of the appropriate pins (refer to [Chapter 12, “General Purpose I/O Module”](#)) prior to configuring the I<sup>2</sup>C module.

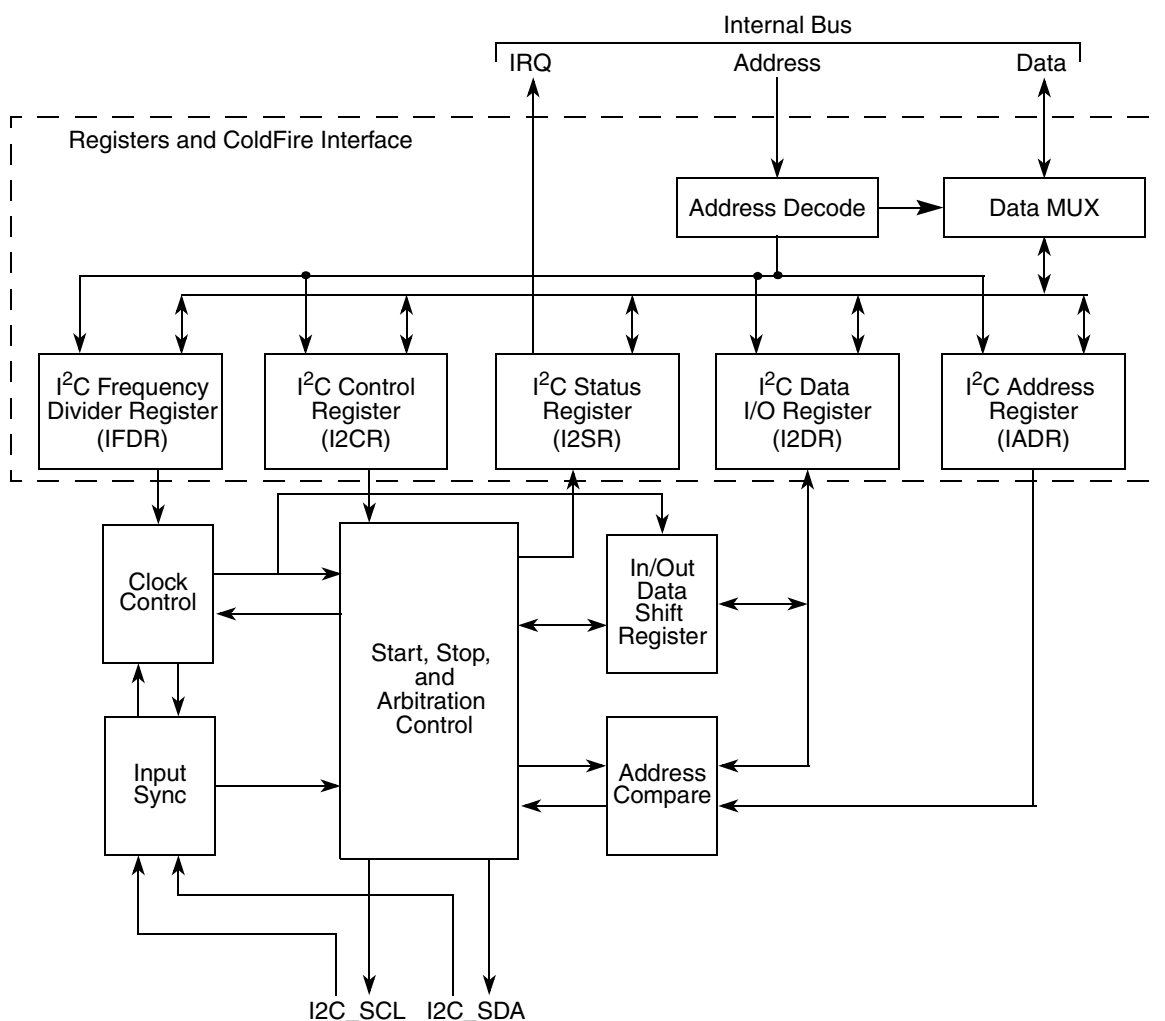
### 26.3 Features

The I<sup>2</sup>C module has the following key features:

- Compatibility with I<sup>2</sup>C bus standard version 2.1
- Support for 3.3-V tolerant devices

- Multiple-master operation
- Software-programmable for one of 50 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

Figure 26-1 is a block diagram of the I<sup>2</sup>C module.



**Figure 26-1. I<sup>2</sup>C Module Block Diagram**

Figure 26-1 shows the relationships of the below I<sup>2</sup>C registers, which are described in Section 26.5, “Memory Map/Register Definition”:

- I<sup>2</sup>C address register (I2ADR)
- I<sup>2</sup>C frequency divider register (I2FDR)
- I<sup>2</sup>C control register (I2CR)
- I<sup>2</sup>C status register (I2SR)
- I<sup>2</sup>C data I/O register (I2DR)

## 26.4 I<sup>2</sup>C System Configuration

The I<sup>2</sup>C module uses a serial data line (I2C\_SDA) and a serial clock line (I2C\_SCL) for data transfer. For I<sup>2</sup>C compliance, all devices connected to these two signals must have open drain or open collector outputs. The logic AND function is exercised on both lines with external pull-up resistors.

Out of reset, the I<sup>2</sup>C default state is as a slave receiver. Thus, when not programmed to be a master or responding to a slave transmit address, the I<sup>2</sup>C module should return to the default slave receiver state. See Section 26.6.1, “Initialization Sequence,” for exceptions.

Normally, a standard communication is composed of four parts: START signal, slave address transmission, data transfer, and STOP signal. These are discussed in the following sections.

### 26.4.1 START Signal

When no other device is bus master (both I2C\_SCL and I2C\_SDA lines are at logic high), a device can initiate communication by sending a START signal (see A in Figure 26-2). A START signal is defined as a high-to-low transition of I2C\_SDA while I2C\_SCL is high. This signal denotes the beginning of a data transfer (each data transfer can be several bytes long) and awakens all slaves.

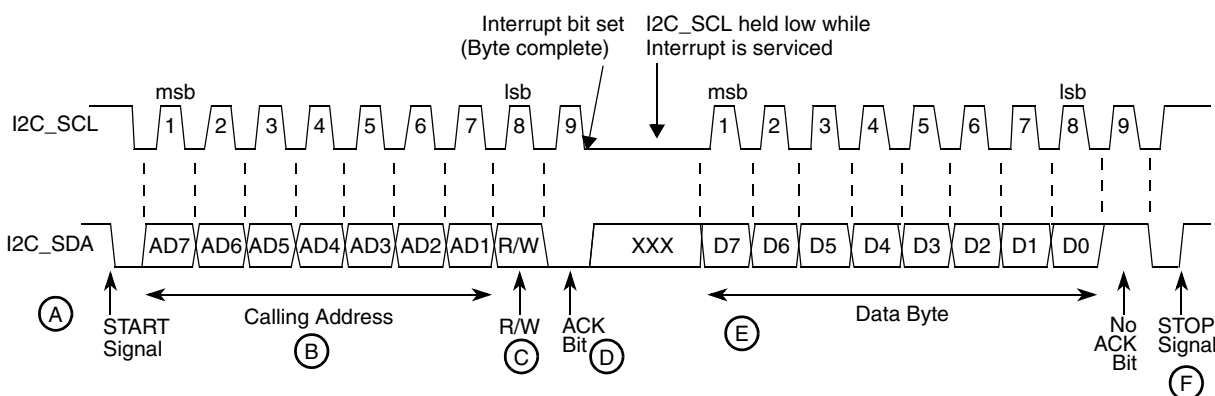


Figure 26-2. I<sup>2</sup>C Standard Communication Protocol

## 26.4.2 Slave Address Transmission

The master sends the slave address in the first byte after the START signal (B). After the seven-bit calling address, it sends the R/W bit (C), which tells the slave data transfer direction (0 = write transfer, 1 = read transfer).

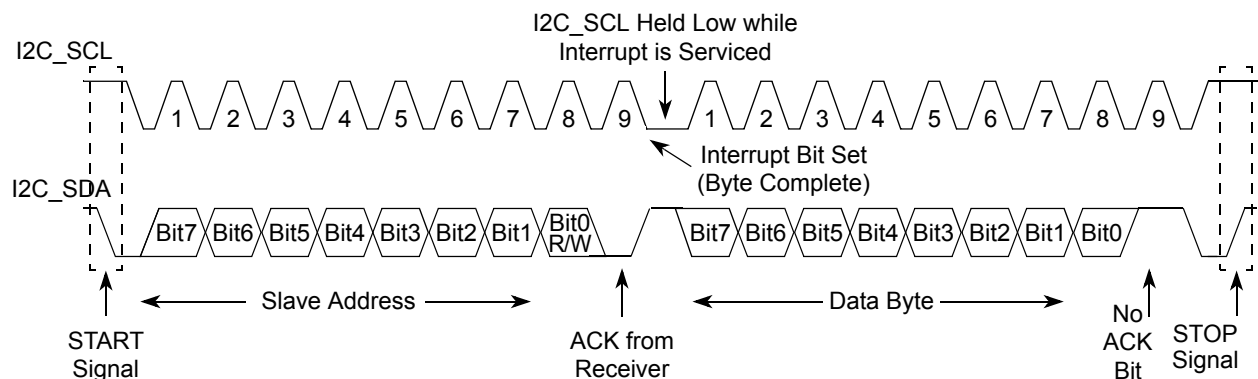
Each slave must have a unique address. An I<sup>2</sup>C master must not transmit its own slave address; it cannot be master and slave at the same time.

The slave whose address matches that sent by the master pulls I2C\_SDA low at the ninth serial clock (D) to return an acknowledge bit.

## 26.4.3 Data Transfer

When successful slave addressing is achieved, the data transfer can proceed (E) on a byte-by-byte basis in the direction specified by the R/W bit sent by the calling master.

Data can be changed only while I2C\_SCL is low and must be held stable while I2C\_SCL is high, as [Figure 26-2](#) shows. I2C\_SCL is pulsed once for each data bit, with the msb being sent first. The receiving device must acknowledge each byte by pulling I2C\_SDA low at the ninth clock; therefore, a data byte transfer takes nine clock pulses. See [Figure 26-3](#).



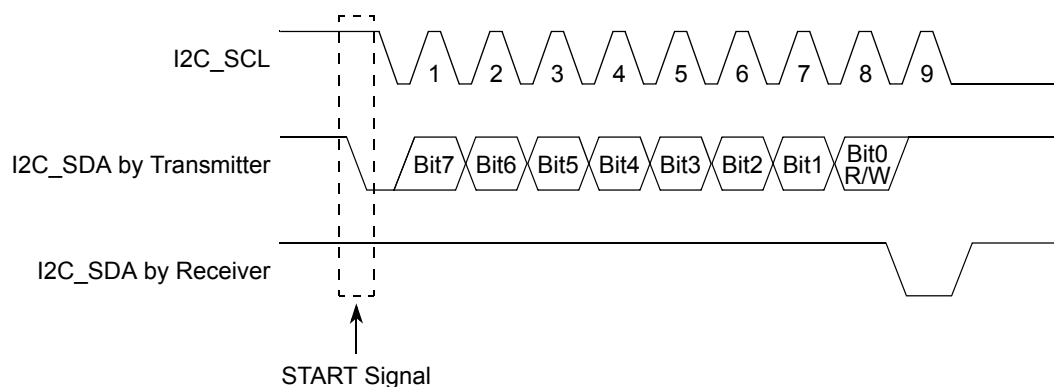
**Figure 26-3. Data Transfer**

## 26.4.4 Acknowledge

The transmitter releases the I2C\_SDA line high during the acknowledge clock pulse as shown in [Figure 26-4](#). The receiver pulls down the I2C\_SDA line during the acknowledge clock pulse so that it remains stable low during the high period of the clock pulse.

If it does not acknowledge the master, the slave receiver must leave I2C\_SDA high. The master can then generate a STOP signal to abort the data transfer or generate a START signal (repeated start, shown in [Figure 26-5](#) and discussed in [Section 26.4.6, “Repeated START”](#)) to start a new calling sequence.



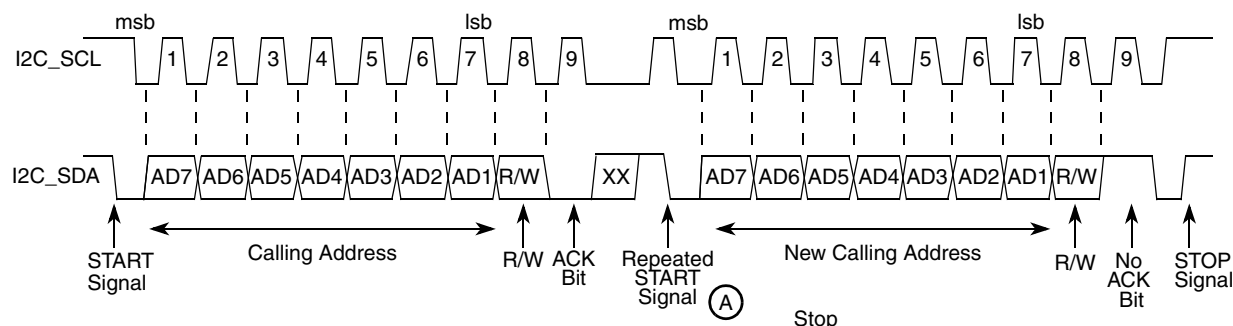


**Figure 26-4. Acknowledgement by Receiver**

If the master receiver does not acknowledge the slave transmitter after a byte transmission, it means end-of-data to the slave. The slave releases I2C\_SDA for the master to generate a STOP or START signal (Figure 26-4).

### 26.4.5 STOP Signal

The master can terminate communication by generating a STOP signal to free the bus. A STOP signal is defined as a low-to-high transition of I2C\_SDA while I2C\_SCL is at logical high (F). The master can generate a STOP even if the slave has generated an acknowledgment at which point the slave must release the bus. The master may also generate a START signal following a calling address, without first generating a STOP signal. Refer to Section 26.4.6, “Repeated START.”



**Figure 26-5. Repeated START**


### 26.4.6 Repeated START


A repeated START signal is a START signal generated without first generating a STOP signal to terminate the communication. The master uses a repeated START to communicate with another slave or with the same slave in a different mode (transmit/receive mode) without releasing the bus.

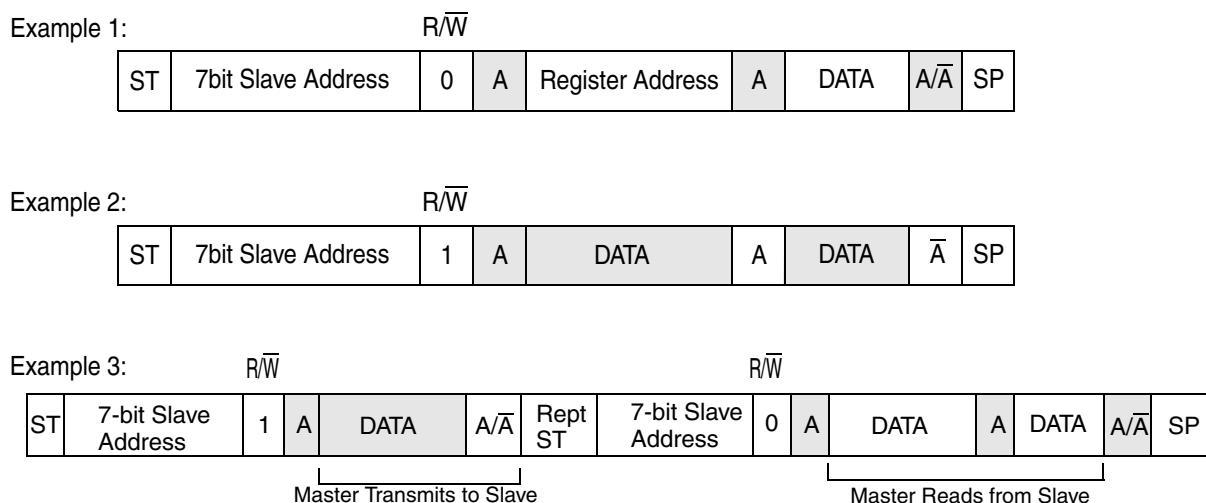
Various combinations of read/write formats are then possible:

- The first example in Figure 26-6 is the case of master-transmitter transmitting to slave-receiver. The transfer direction is not changed.
- The second example in Figure 26-6 is the master reading the slave immediately after the first byte. At the moment of the first acknowledge, the master-transmitter becomes a master-receiver and the slave-receiver becomes slave-transmitter.
- In the third example in Figure 26-6 the START condition and slave address are both repeated using the repeated START signal. This is to communicate with same slave in a different mode without releasing the bus. The master transmits data to the slave first, and then the master reads data from slave by reversing the R/W bit.

ST = Start  
 SP = Stop  
 A = Acknowledge (I2C\_SDA low)  
 A = Not Acknowledge (I2C\_SDA high)  
 Rept ST = Repeated Start

 From Master to Slave

 From Slave to Master



### Figure 26-6. Data Transfer, Combined Format

<sup>1</sup> Note: No acknowledge on the last byte

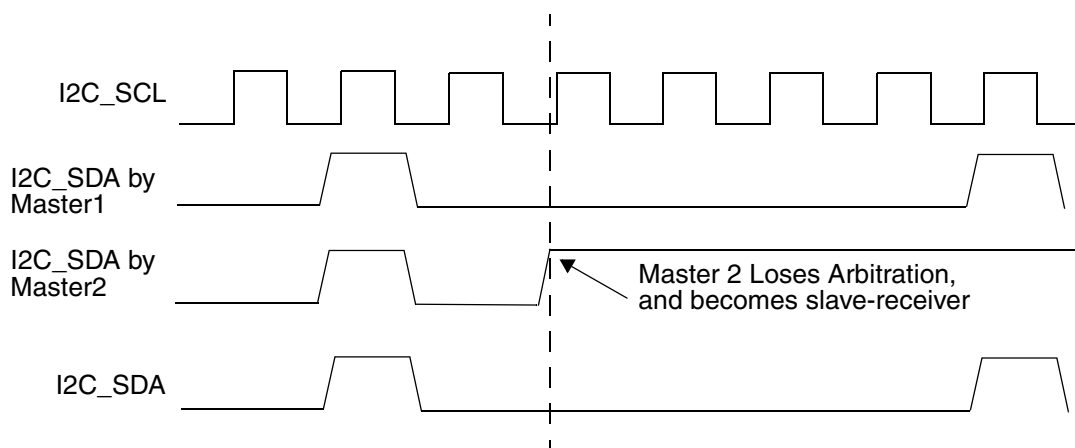
### 26.4.7 Clock Synchronization and Arbitration

I<sup>2</sup>C is a true multi-master bus that allows more than one master to be connected to it. If two or more masters devices simultaneously request control of the bus, a clock synchronization procedure determines the bus clock. Because wire-AND logic is performed on the I2C\_SCL line, a high-to-low transition on the I2C\_SCL line affects all the devices connected on the bus. The devices start counting their low period and once a device's clock has gone low, it holds the I2C\_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 I2C\_SCL line if another device clock is still within

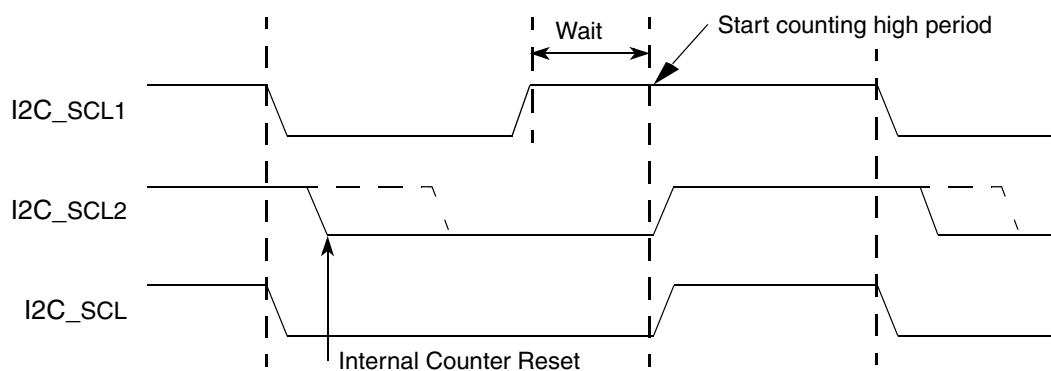
its low period. Therefore, synchronized clock I2C\_SCL is held low by the device with the longest low period.

Devices with shorter low periods enter a high wait state during this time (see Figure 26-8). When all devices concerned have counted off their low period, the synchronized clock I2C\_SCL line is released and pulled high. There is then no difference between the device clocks and the state of the I2C\_SCL line and all the devices start counting their high periods. The first device to complete its high period pulls the I2C\_SCL line low again.

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 over to slave receive mode and stop driving I2C\_SDA output (see Figure 26-7). In this case the transition from master to slave mode does not generate a STOP condition. Meanwhile, hardware sets I2SR[IAL] to indicate loss of arbitration.



**Figure 26-7. Arbitration Procedure**



**Figure 26-8. Clock Synchronization**

## 26.4.8 Handshaking and Clock Stretching

The clock synchronization mechanism can be used as a handshake in data transfers. Slave devices can hold I2C\_SCL low after completing one byte transfer. In such a case, the clock mechanism halts the bus clock and forces the master clock into wait states until the slave releases I2C\_SCL.

Slaves may also slow down the transfer bit rate. After the master has driven I2C\_SCL low, the slave can drive I2C\_SCL low for the required period and then release it. If the slave I2C\_SCL low period is longer than the master I2C\_SCL low period, the resulting I2C\_SCL bus signal low period is stretched.

## 26.5 Memory Map/Register Definition

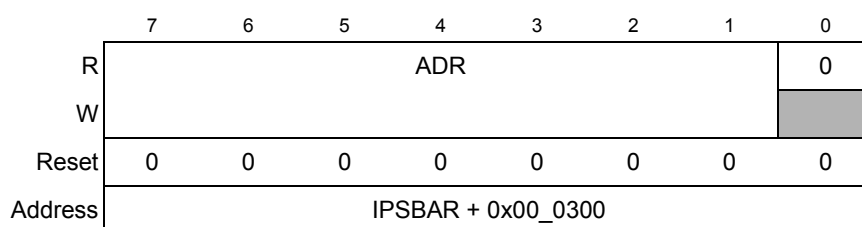
Table 26-1 lists the configuration registers used in the I<sup>2</sup>C interface.

**Table 26-1. I<sup>2</sup>C Interface Memory Map**

IPSBAR Offset	Mnemonic	[31:24]	[23:0]	Access
0x0300	I2ADR	I <sup>2</sup> C Address Register	Reserved	R/W
0x0304	I2FDR	I <sup>2</sup> C Frequency Divider Register	Reserved	R/W
0x0308	I2CR	I <sup>2</sup> C Control Register	Reserved	R/W
0x030C	I2SR	I <sup>2</sup> C Status Register	Reserved	R/W
0x0310	I2DR	I <sup>2</sup> C Data I/O Register	Reserved	R/W

### 26.5.1 I<sup>2</sup>C Address Register (I2ADR)

The I2ADR holds the address the I<sup>2</sup>C responds to when addressed as a slave. Note that it is not the address sent on the bus during the address transfer.



**Figure 26-9. I<sup>2</sup>C Address Register (I2ADR)**

**Table 26-2. I2ADR Field Descriptions**

Bits	Name	Description
7–1	ADR	Slave address. Contains the specific slave address to be used by the I <sup>2</sup> C module. Slave mode is the default I <sup>2</sup> C mode for an address match on the bus.
0	—	Reserved, should be cleared.

## 26.5.2 I<sup>2</sup>C Frequency Divider Register (I2FDR)

The I2FDR, shown in [Figure 26-10](#), provides a programmable prescaler to configure the I<sup>2</sup>C clock for bit-rate selection.

	7	6	5	4	3	2	1	0
R	0	0	IC					
W								
Reset	0	0	0	0	0	0	0	0
Address	IPSBAR + 0x00_0304							

**Figure 26-10. I<sup>2</sup>C Frequency Divider Register (I2FDR)**

**Table 26-3. I2FDR Field Descriptions**

Bits	Name	Description																																																																																																																																								
7–6	—	Reserved, should be cleared.																																																																																																																																								
5–0	IC	<div>I<sup>2</sup>C clock rate. Prescales the clock for bit-rate selection. The serial bit clock frequency is equal to the internal bus clock divided by the divider shown below. Due to potentially slow I2C_SCL and I2C_SDA rise and fall times, bus signals are sampled at the prescaler frequency.</div> <table><tr><th>IC</th><th>Divider</th><th>IC</th><th>Divider</th><th>IC</th><th>Divider</th><th>IC</th><th>Divider</th></tr><tr><td>0x00</td><td>28</td><td>0x10</td><td>288</td><td>0x20</td><td>20</td><td>0x30</td><td>160</td></tr><tr><td>0x01</td><td>30</td><td>0x11</td><td>320</td><td>0x21</td><td>22</td><td>0x31</td><td>192</td></tr><tr><td>0x02</td><td>34</td><td>0x12</td><td>384</td><td>0x22</td><td>24</td><td>0x32</td><td>224</td></tr><tr><td>0x03</td><td>40</td><td>0x13</td><td>480</td><td>0x23</td><td>26</td><td>0x33</td><td>256</td></tr><tr><td>0x04</td><td>44</td><td>0x14</td><td>576</td><td>0x24</td><td>28</td><td>0x34</td><td>320</td></tr><tr><td>0x05</td><td>48</td><td>0x15</td><td>640</td><td>0x25</td><td>32</td><td>0x35</td><td>384</td></tr><tr><td>0x06</td><td>56</td><td>0x16</td><td>768</td><td>0x26</td><td>36</td><td>0x36</td><td>448</td></tr><tr><td>0x07</td><td>68</td><td>0x17</td><td>960</td><td>0x27</td><td>40</td><td>0x37</td><td>512</td></tr><tr><td>0x08</td><td>80</td><td>0x18</td><td>1152</td><td>0x28</td><td>48</td><td>0x38</td><td>640</td></tr><tr><td>0x09</td><td>88</td><td>0x19</td><td>1280</td><td>0x29</td><td>56</td><td>0x39</td><td>768</td></tr><tr><td>0x0A</td><td>104</td><td>0x1A</td><td>1536</td><td>0x2A</td><td>64</td><td>0x3A</td><td>896</td></tr><tr><td>0x0B</td><td>128</td><td>0x1B</td><td>1920</td><td>0x2B</td><td>72</td><td>0x3B</td><td>1024</td></tr><tr><td>0x0C</td><td>144</td><td>0x1C</td><td>2304</td><td>0x2C</td><td>80</td><td>0x3C</td><td>1280</td></tr><tr><td>0x0D</td><td>160</td><td>0x1D</td><td>2560</td><td>0x2D</td><td>96</td><td>0x3D</td><td>1536</td></tr><tr><td>0x0E</td><td>192</td><td>0x1E</td><td>3072</td><td>0x2E</td><td>112</td><td>0x3E</td><td>1792</td></tr><tr><td>0x0F</td><td>240</td><td>0x1F</td><td>3840</td><td>0x2F</td><td>128</td><td>0x3F</td><td>2048</td></tr></table>	IC	Divider	IC	Divider	IC	Divider	IC	Divider	0x00	28	0x10	288	0x20	20	0x30	160	0x01	30	0x11	320	0x21	22	0x31	192	0x02	34	0x12	384	0x22	24	0x32	224	0x03	40	0x13	480	0x23	26	0x33	256	0x04	44	0x14	576	0x24	28	0x34	320	0x05	48	0x15	640	0x25	32	0x35	384	0x06	56	0x16	768	0x26	36	0x36	448	0x07	68	0x17	960	0x27	40	0x37	512	0x08	80	0x18	1152	0x28	48	0x38	640	0x09	88	0x19	1280	0x29	56	0x39	768	0x0A	104	0x1A	1536	0x2A	64	0x3A	896	0x0B	128	0x1B	1920	0x2B	72	0x3B	1024	0x0C	144	0x1C	2304	0x2C	80	0x3C	1280	0x0D	160	0x1D	2560	0x2D	96	0x3D	1536	0x0E	192	0x1E	3072	0x2E	112	0x3E	1792	0x0F	240	0x1F	3840	0x2F	128	0x3F	2048
IC	Divider	IC	Divider	IC	Divider	IC	Divider																																																																																																																																			
0x00	28	0x10	288	0x20	20	0x30	160																																																																																																																																			
0x01	30	0x11	320	0x21	22	0x31	192																																																																																																																																			
0x02	34	0x12	384	0x22	24	0x32	224																																																																																																																																			
0x03	40	0x13	480	0x23	26	0x33	256																																																																																																																																			
0x04	44	0x14	576	0x24	28	0x34	320																																																																																																																																			
0x05	48	0x15	640	0x25	32	0x35	384																																																																																																																																			
0x06	56	0x16	768	0x26	36	0x36	448																																																																																																																																			
0x07	68	0x17	960	0x27	40	0x37	512																																																																																																																																			
0x08	80	0x18	1152	0x28	48	0x38	640																																																																																																																																			
0x09	88	0x19	1280	0x29	56	0x39	768																																																																																																																																			
0x0A	104	0x1A	1536	0x2A	64	0x3A	896																																																																																																																																			
0x0B	128	0x1B	1920	0x2B	72	0x3B	1024																																																																																																																																			
0x0C	144	0x1C	2304	0x2C	80	0x3C	1280																																																																																																																																			
0x0D	160	0x1D	2560	0x2D	96	0x3D	1536																																																																																																																																			
0x0E	192	0x1E	3072	0x2E	112	0x3E	1792																																																																																																																																			
0x0F	240	0x1F	3840	0x2F	128	0x3F	2048																																																																																																																																			

## 26.5.3 I<sup>2</sup>C Control Register (I2CR)

The I2CR is used to enable the I<sup>2</sup>C module and the I<sup>2</sup>C interrupt. It also contains bits that govern operation as a slave or a master.

	7	6	5	4	3	2	1	0
R	IEN	IEN	MSTA	MTX	TXAK	RSTA	0	0
W								
Reset	0	0	0	0	0	0	0	0
Address	IPSBAR + 0x00_0308							

**Figure 26-11. I<sup>2</sup>C Control Register (I2CR)**

**Table 26-4. I2CR Field Descriptions**

Bits	Name	Description
7	IEN	I <sup>2</sup> C enable. Controls the software reset of the entire I <sup>2</sup> C module. If the module is enabled in the middle of a byte transfer, slave mode ignores the current bus transfer and starts operating when the next START condition is detected. Master mode is not aware that the bus is busy; so initiating a start cycle may corrupt the current bus cycle, ultimately causing either the current master or the I <sup>2</sup> C module to lose arbitration, after which bus operation returns to normal. 0 The I <sup>2</sup> C module is disabled, but registers can still be accessed. 1 The I <sup>2</sup> C module is enabled. This bit must be set before any other I2CR bits have any effect.
6	IEN	I <sup>2</sup> C interrupt enable. 0 I <sup>2</sup> C module interrupts are disabled, but currently pending interrupt condition are not cleared. 1 I <sup>2</sup> C module interrupts are enabled. An I <sup>2</sup> C interrupt occurs if I2SR[IIF] is also set.
5	MSTA	Master/slave mode select bit. If the master loses arbitration, MSTA is cleared without generating a STOP signal. 0 Slave mode. Changing MSTA from 1 to 0 generates a STOP and selects slave mode. 1 Master mode. Changing MSTA from 0 to 1 signals a START on the bus and selects master mode.
4	MTX	Transmit/receive mode select bit. Selects the direction of master and slave transfers. 0 Receive 1 Transmit. When the device is addressed as a slave, software should set MTX according to I2SR[SRW]. In master mode, MTX should be set according to the type of transfer required. Therefore, when the MCU addresses a slave device, MTX is always 1.
3	TXAK	Transmit acknowledge enable. Specifies the value driven onto I2C_SDA during acknowledge cycles for both master and slave receivers. Note that writing TXAK applies only when the I <sup>2</sup> C bus is a receiver. 0 An acknowledge signal is sent to the bus at the ninth clock bit after receiving one byte of data. 1 No acknowledge signal response is sent (that is, acknowledge bit = 1).

**Table 26-4. I2CR Field Descriptions (Continued)**

Bits	Name	Description
2	RSTA	Repeat start. Always read as 0. Attempting a repeat start without bus mastership causes loss of arbitration. 0 No repeat start 1 Generates a repeated START condition.
1–0	—	Reserved, should be cleared.

## 26.5.4 I<sup>2</sup>C Status Register (I2SR)

This I2SR contains bits that indicate transaction direction and status.

	7	6	5	4	3	2	1	0
R	ICF	IAAS	IBB	IAL	0	SRW	IIF	RXAK
W								
Reset	1	0	0	0	0	0	0	1
Address	IPSBAR + 0x00_030C							

**Figure 26-12. I<sup>2</sup>C Status Register (I2SR)****Table 26-5. I2SR Field Descriptions**

Bits	Name	Description
7	ICF	Data transferring bit. While one byte of data is transferred, ICF is cleared. 0 Transfer in progress 1 Transfer complete. Set by the falling edge of the ninth clock of a byte transfer.
6	IAAS	I <sup>2</sup> C addressed as a slave bit. The CPU is interrupted if I2CR[IIFEN] is set. Next, the CPU must check SRW and set its TX/RX mode accordingly. Writing to I2CR clears this bit. 0 Not addressed. 1 Addressed as a slave. Set when its own address (IADR) matches the calling address.
5	IBB	I <sup>2</sup> C bus busy bit. Indicates the status of the bus. 0 Bus is idle. If a STOP signal is detected, IBB is cleared. 1 Bus is busy. When START is detected, IBB is set.
4	IAL	Arbitration lost. Set by hardware in the following circumstances. (IAL must be cleared by software by writing zero to it.) <ul style="list-style-type: none"> <li>I2C_SDA sampled low when the master drives high during an address or data-transmit cycle.</li> <li>I2C_SDA sampled low when the master drives high during the acknowledge bit of a data-receive cycle.</li> <li>A start cycle is attempted when the bus is busy.</li> <li>A repeated start cycle is requested in slave mode.</li> <li>A stop condition is detected when the master did not request it.</li> </ul>
3	—	Reserved, should be cleared.

**Table 26-5. I2SR Field Descriptions (Continued)**

Bits	Name	Description
2	SRW	Slave read/write. When IAAS is set, SRW indicates the value of the R/W command bit of the calling address sent from the master. SRW is valid only when a complete transfer has occurred, no other transfers have been initiated, and the I <sup>2</sup> C module is a slave and has an address match. 0 Slave receive, master writing to slave. 1 Slave transmit, master reading from slave.
1	IIF	I <sup>2</sup> C interrupt. Must be cleared by software by writing a zero in the interrupt routine. 0 No I <sup>2</sup> C interrupt pending 1 An interrupt is pending, which causes a processor interrupt request (if ILEN = 1). Set when one of the following occurs: <ul style="list-style-type: none"> <li>• Complete one byte transfer (set at the falling edge of the ninth clock)</li> <li>• Reception of a calling address that matches its own specific address in slave-receive mode</li> <li>• Arbitration lost</li> </ul>
0	RXAK	Received acknowledge. The value of I2C_SDA during the acknowledge bit of a bus cycle. 0 An acknowledge signal was received after the completion of 8-bit data transmission on the bus 1 No acknowledge signal was detected at the ninth clock.

### 26.5.5 I<sup>2</sup>C Data I/O Register (I2DR)

In master-receive mode, reading the I2DR allows a read to occur and initiates next the byte data to be received. In slave mode, the same function is available once the I<sup>2</sup>C has received its slave address.

	7	6	5	4	3	2	1	0
R	Data							
W								
Reset	0	0	0	0	0	0	0	0
Address	IPSBAR + 0x00_0310							

**Figure 26-13. I<sup>2</sup>C Data I/O Register (I2DR)**



**Table 26-6. I2DR Field Description**

Bit	Name	Description
7–0	DATA	<p>I<sup>2</sup>C data. In master transmit mode, when data is written to this register, a data transfer is initiated. The most significant bit is sent first. In master receive mode, reading this register initiates the reception of the next byte of data. In slave mode, the same functions are available after an address match has occurred.</p> <p><b>Note:</b> 1. In master transmit mode, the first byte of data written to I2DR following assertion of I2CR[MSTA] is used for the address transfer and should comprise the calling address (in position D7–D1) concatenated with the required R/W bit (in position D0). This bit (D0) is not automatically appended by the hardware, software must provide the appropriate R/W bit.</p> <p><b>Note:</b> 2. I2CR[MSTA] generates a start when a master does not already own the bus. I2CR[RSTA] generates a start (restart) without the master first issuing a stop (i.e., the master already owns the bus). In order to start the read of data, a dummy read to this register starts the read process from the slave. The next read of the I2DR register contains the actual data.</p>

## 26.6 I<sup>2</sup>C Programming Examples

The following examples show programming for initialization, signaling START, post-transfer software response, signalling STOP, and generating a repeated START.

### 26.6.1 Initialization Sequence

Before the interface can transfer serial data, registers must be initialized, as follows:

1. Set I2FDR[IC] to obtain I2C\_SCL frequency from the system bus clock. See [Section 26.5.2, “I2C Frequency Divider Register \(I2FDR\).”](#)
2. Update the I2ADR to define its slave address.
3. Set I2CR[IEN] to enable the I<sup>2</sup>C bus interface system.
4. Modify the I2CR to select or deselect master/slave mode, transmit/receive mode, and interrupt-enable or not.

#### NOTE

If I2SR[IBB] is set when the I<sup>2</sup>C bus module is enabled, execute the following pseudocode sequence before proceeding with normal initialization code. This issues a STOP command to the slave device, placing it in idle state as if it were just power-cycled on.

```

I2CR = 0x0
I2CR = 0xA0
dummy read of I2DR
I2SR = 0x0
I2CR = 0x0

```

## 26.6.2 Generation of START

After completion of the initialization procedure, serial data can be transmitted by selecting the master transmitter mode. On a multiple-master bus system, I2SR[IBB] must be tested to determine whether the serial bus is free. If the bus is free (IBB = 0), the START signal and the first byte (the slave address) can be sent. The data written to the data register comprises the address of the desired slave and the lsb indicates the transfer direction.

The free time between a STOP and the next START condition is built into the hardware that generates the START cycle. Depending on the relative frequencies of the system clock and the I2C\_SCL period, it may be necessary to wait until the I<sup>2</sup>C is busy after writing the calling address to the I2DR before proceeding with the following instructions.

The following example signals START and transmits the first byte of data (slave address):

```
CHFLAG  MOVE.B I2SR,-(A0)                ;Check I2SR[MBB]
        BTST.B #5, (A0)+
        BNE.S CHFLAG                    ;If I2SR[MBB] = 1, wait until it is clear
TXSTART  MOVE.B I2CR,-(A0)                ;Set transmit mode
        BSET.B #4, (A0)
        MOVE.B (A0)+, I2CR
        MOVE.B I2CR, -(A0)                ;Set master mode
        BSET.B #5, (A0)                  ;Generate START condition
        MOVE.B (A0)+, I2CR
        MOVE.B CALLING,-(A0)              ;Transmit the calling address, D0=R/W
        MOVE.B (A0)+, I2DR
IFREE    MOVE.B I2SR,-(A0)                ;Check I2SR[MBB]
        ;If it is clear, wait until it is set.
        BTST.B #5, (A0)+
        BEQ.S IFREE;
```

## 26.6.3 Post-Transfer Software Response

Sending or receiving a byte sets the I2SR[ICF], which indicates one byte communication is finished. I2SR[IIF] is also set. An interrupt is generated if the interrupt function is enabled during initialization by setting I2CR[IEN]. Software must first clear I2SR[IIF] in the interrupt routine. I2SR[ICF] is cleared either by reading from I2DR in receive mode or by writing to I2DR in transmit mode.

Software can service the I<sup>2</sup>C I/O in the main program by monitoring IIF if the interrupt function is disabled. Polling should monitor IIF rather than ICF because that 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; that is, the address is sent. If master receive mode is required I2CR[MTX] should be toggled.

During slave-mode address cycles (I2SR[IAAS] = 1), I2SR[SRW] is read to determine the direction of the next transfer. MTX is programmed accordingly. For slave-mode data cycles (IAAS = 0), SRW is invalid. MTX should be read to determine the current transfer direction.

The following is an example of a software response by a master transmitter in the interrupt routine (see Figure 26-14).

```

I2SR      LEA.L I2SR, -(A7)           ;Load effective address
          BCLR.B #1, (A7)+           ;Clear the IIF flag
          MOVE.B I2CR, -(A7)         ;Push the address on stack,
          BTST.B #5, (A7)+           ;check the MSTA flag
          BEQ.S SLAVE                ;Branch if slave mode
          MOVE.B I2CR, -(A7)         ;Push the address on stack
          BTST.B #4, (A7)+           ;check the mode flag
          BEQ.S RECEIVE              ;Branch if in receive mode
          MOVE.B I2SR, -(A7)         ;Push the address on stack,
          BTST.B #0, (A7)+           ;check ACK from receiver
          BNE.B END                  ;If no ACK, end of transmission
TRANSMIT  MOVE.B DATABUF, -(A7)      ;Stack data byte
          MOVE.B (A7)+, I2DR         ;Transmit next byte of data

```

## 26.6.4 Generation of STOP

A data transfer ends when the master signals a STOP, which can occur after all data is sent, as in the following example.

```

MASTX     MOVE.B I2SR, -(A7)          ;If no ACK, branch to end
          BTST.B #0, (A7)+
          BNE.B END
          MOVE.B TXCNT, D0             ;Get value from the transmitting counter
          BEQ.S END                   ;If no more data, branch to end
          MOVE.B DATABUF, -(A7)       ;Transmit next byte of data
          MOVE.B (A7)+, I2DR
          MOVE.B TXCNT, D0             ;Decrease the TXCNT
          SUBQ.L #1, D0
          MOVE.B D0, TXCNT
          BRA.S EMASTX;Exit
END        LEA.L I2CR, -(A7)          ;Generate a STOP condition
          BCLR.B #5, (A7)+
          EMASTX RTE                  ;Return from interrupt

```

For a master receiver to terminate a data transfer, it must inform the slave transmitter by not acknowledging the last data byte. This is done by setting I2CR[TXAK] before reading the next-to-last byte. Before the last byte is read, a STOP signal must be generated, as in the following example.

```

MASR      MOVE.B RXCNT, D0            ;Decrease RXCNT
          SUBQ.L #1, D0
          MOVE.B D0, RXCNT
          BEQ.S ENMASR                ;Last byte to be read
          MOVE.B RXCNT, D1            ;Check second-to-last byte to be read

```

```

EXTB.L D1
SUBI.L #1,D1;
BNE.S NXMAR ;Not last one or second last
LAMAR BSET.B #3,I2CR ;Disable ACK
BRA NXMAR

ENMASR BCLR.B #5,I2CR ;Last one, generate STOP signal
NXMAR MOVE.B I2DR,RXBUF ;Read data and store RTE

```

## 26.6.5 Generation of Repeated START

After the data transfer, if the master still wants the bus, it can signal another START followed by another slave address without signalling a STOP, as in the following example.

```

RESTART MOVE.B I2CR,-(A7) ;Repeat START (RESTART)
BSET.B #2,(A7)
MOVE.B (A7)+, I2CR
MOVE.B CALLING,-(A7) ;Transmit the calling address, D0=R/W-
MOVE.B CALLING,-(A7)

MOVE.B (A7)+, I2DR

```

## 26.6.6 Slave Mode

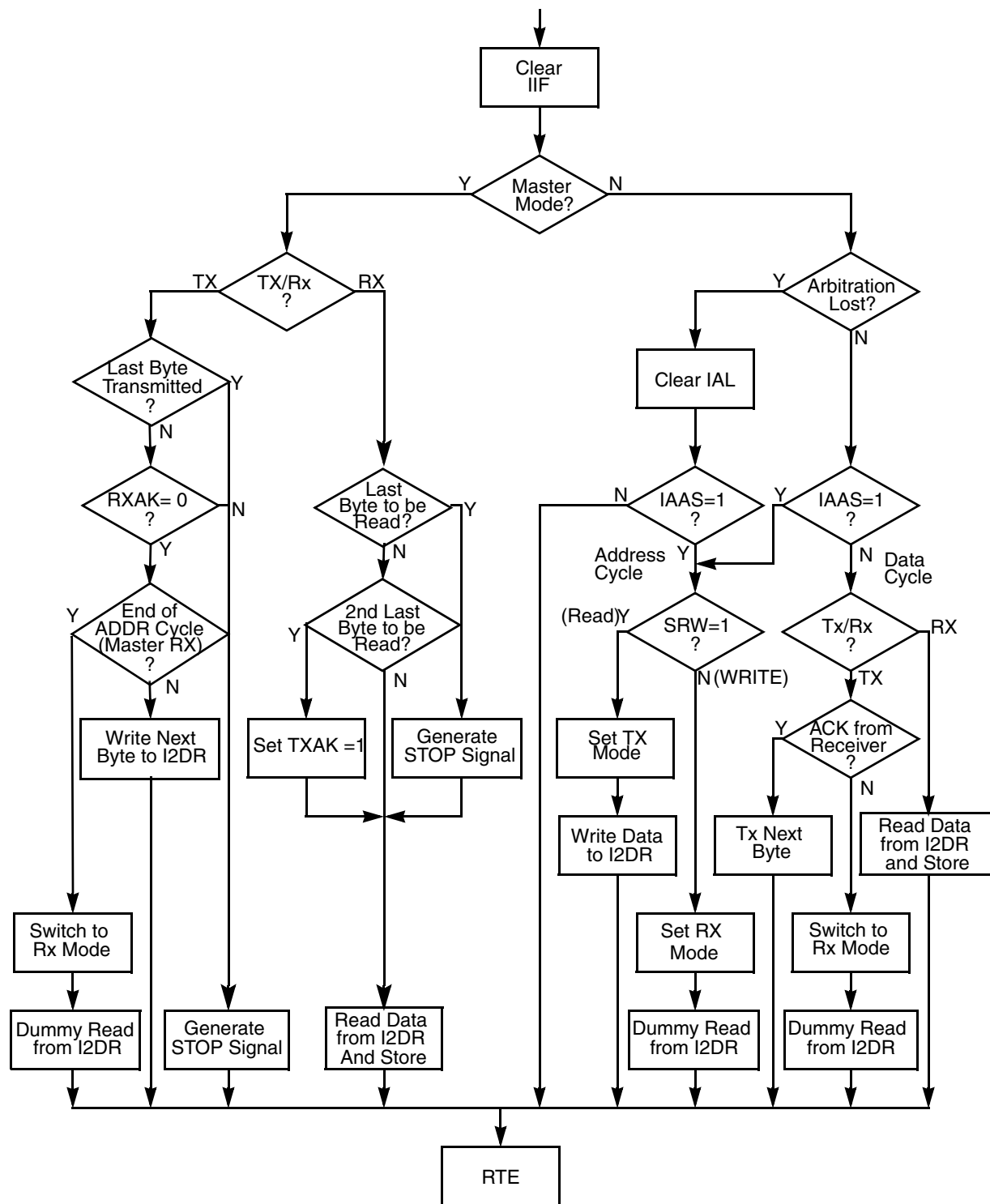
In the slave interrupt service routine, software should poll the I2SR[IAAS] bit to determine if the controller has received its slave address. If IAAS is set, software should set the transmit/receive mode select bit (I2CR[MTX]) according to the I2SR[SRW]. Writing to the I2CR clears the IAAS automatically. The only time IAAS is read as set is from the interrupt at the end of the address cycle where an address match occurred; interrupts resulting from subsequent data transfers will have IAAS cleared. A data transfer can now be initiated by writing information to I2DR for slave transmits, or read from I2DR in slave-receive mode. A dummy read of I2DR in slave/receive mode releases I2C\_SCL, allowing the master to send data.

In the slave transmitter routine, I2SR[RXAK] must be tested before sending the next byte of data. Setting RXAK means an end-of-data signal from the master receiver, after which software must switch it from transmitter to receiver mode. Reading I2DR then releases I2C\_SCL so that the master can generate a STOP signal.

## 26.6.7 Arbitration Lost

If several devices try to engage the bus at the same time, one becomes master. Hardware immediately switches devices that lose arbitration to slave receive mode. Data output to I2C\_SDA stops, but I2C\_SCL is still generated until the end of the byte during which arbitration is lost. An interrupt occurs at the falling edge of the ninth clock of this transfer with I2SR[IAL] = 1 and I2CR[MSTA] = 0.

If a device that is not a master tries to transmit or do a START, hardware inhibits the transmission, clears MSTA without signalling a STOP, generates an interrupt to the CPU, and sets IAL to indicate a failed attempt to engage the bus. When considering these cases, the slave service routine should first test IAL and software should clear it if it is set.

Figure 26-14. Flow-Chart of Typical I<sup>2</sup>C Interrupt Routine

# Chapter 27

## UART Modules

### 27.1 Introduction

This chapter describes the use of the universal asynchronous receiver/transmitters (UARTs) implemented on the MCF5275 and includes programming examples.

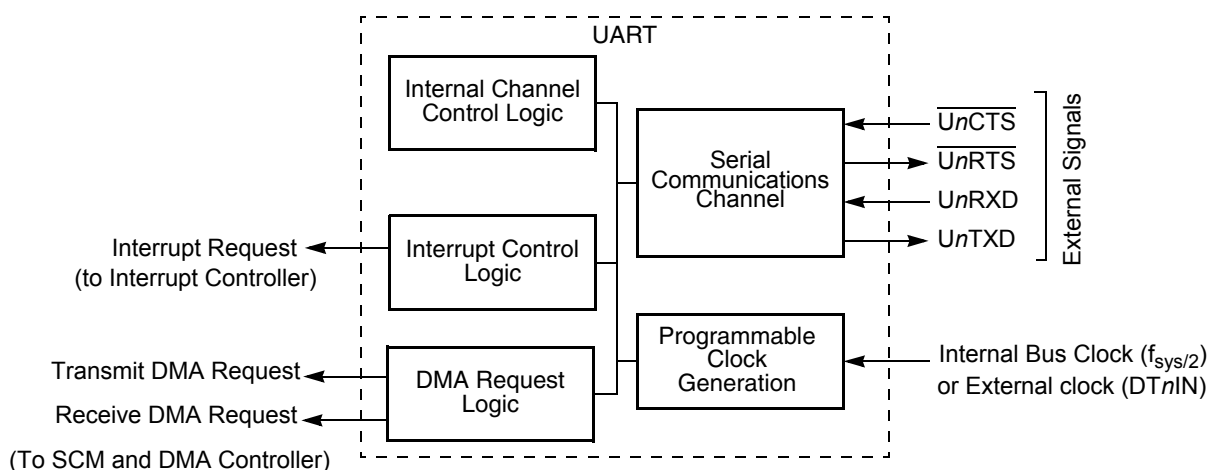
#### NOTE

The designation “*n*” is used throughout this section to refer to registers or signals associated with one of the three identical UART modules: UART0, UART1, or UART2.

#### 27.1.1 Overview

The MCF5275 contains three independent UARTs. Each UART can be clocked by the internal bus clock, eliminating the need for an external UART clock. As [Figure 27-1](#) shows, each UART module interfaces directly to the CPU and consists of the following:

- Serial communication channel
- Programmable clock generation
- Interrupt control logic and DMA request logic
- Internal channel control logic



**Figure 27-1. UART Block Diagram**

**NOTE**

UART $n$  can be clocked by the DT $n$ IN pin. However, if the timers are used, then input capture mode is not available for that timer.

The serial communication channel provides a full-duplex asynchronous/synchronous receiver and transmitter deriving an operating frequency from the internal bus clock or an external clock using the timer pin. The transmitter converts parallel data from the CPU to a serial bit stream, inserting appropriate start, stop, and parity bits. It outputs the resulting stream on the channel transmitter serial data output (UnTXD). See [Section 27.4.2.1, “Transmitter.”](#)

The receiver converts serial data from the channel receiver serial data input (UnRXD) to parallel format, checks for a start, stop, and parity bits, or break conditions, and transfers the assembled character onto the bus during read operations. The receiver may be polled, interrupt driven, or use DMA requests for servicing. See [Section 27.4.2.2, “Receiver.”](#)

**NOTE**

The GPIO module must be configured to enable the peripheral function of the appropriate pins (refer to [Chapter 12, “General Purpose I/O Module”](#)) prior to configuring the UART module.

## 27.1.2 Features

The MCF5275 contains three independent UART modules with the following features:

- Each can be clocked by an external clock or by the internal bus clock (eliminating a need for an external UART clock).
- Full-duplex asynchronous/synchronous receiver/transmitter channel
- Quadruple-buffered receiver
- Double-buffered transmitter
- Independently programmable receiver and transmitter clock sources
- Programmable data format:
  - 5–8 data bits plus parity
  - Odd, even, no parity, or force parity
  - One, one-and-a-half, or two stop bits
- Each channel programmable to normal (full-duplex), automatic echo, local loop-back, or remote loop-back mode
- Automatic wake-up mode for multidrop applications
- Four maskable interrupt conditions
- All three UARTs have DMA request capability
- Parity, framing, and overrun error detection



- False-start bit detection
- Line-break detection and generation
- Detection of breaks originating in the middle of a character
- Start/end break interrupt/status

## 27.2 External Signal Description

Figure 27-1 shows both the external and internal signal groups.

An internal interrupt request signal is provided to notify the interrupt controller of an interrupt condition. The output is the logical NOR of unmasked  $UISR_n$  bits. The interrupt level and priority are programmed in the interrupt controller—ICR13 for UART0, ICR14 for UART1, and ICR15 for UART2. See Section 13.2.1.6, “Interrupt Control Register (ICR $_n$ , (x = 1, 2,..., 63)).”

Note that the UARTs can also be configured to automatically transfer data by using the DMA rather than interrupting the core. When there is data in the receiver FIFO or when the transmit holding register is empty, a DMA request can be issued. For more information on generating DMA requests, refer to Section 27.4.6.1.2, “Setting up the UART to Request DMA Service,” and Section 18.3.1, “DMA Request Control (DMAREQC).”

Table 27-1 briefly describes the UART module signals.

### NOTE

The terms ‘assertion’ and ‘negation’ are used to avoid confusion between active-low and active-high signals. ‘Asserted’ indicates that a signal is active, independent of the voltage level; ‘negated’ indicates that a signal is inactive.

**Table 27-1. UART Module Signals**

Signal	Description
Transmitter Serial Data Output (UnTXD)	UnTXD is held high (mark condition) when the transmitter is disabled, idle, or operating in the local loop-back mode. Data is shifted out on UnTXD on the falling edge of the clock source, with the least significant bit (lsb) sent first.
Receiver Serial Data Input (UnRXD)	Data received on UnRXD is sampled on the rising edge of the clock source, with the lsb received first.
Clear-to-Send (UnCTS)	This input can generate an interrupt on a change of state.
Request-to-Send (UnRTS)	This output can be programmed to be negated or asserted automatically by either the receiver or the transmitter. When connected to a transmitter's UnCTS, UnRTS can control serial data flow.

Figure 27-2 shows a signal configuration for a UART/RS-232 interface.

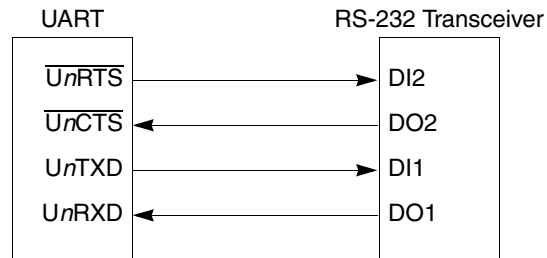


Figure 27-2. UART/RS-232 Interface

## 27.3 Memory Map/Register Definition

This section contains a detailed description of each register and its specific function. Flowcharts in [Section 27.4.6, “Programming,”](#) describe basic UART module programming. The operation of the UART module is controlled by writing control bytes into the appropriate registers. [Table 27-2](#) is a memory map for UART module registers.

### NOTE

UART registers are accessible only as bytes.

### NOTE

Interrupt can mean either an interrupt request asserted to the CPU or a DMA request.

Table 27-2. UART Module Memory Map

IPSBAR Offset	[31:24]	[23:16]	[15:8]	[7:0]
UART0 UART1 UART2				
0x00_0200 0x00_0240 0x00_0280	UART Mode Registers <sup>1</sup> (UMR1n) , (UMR2n)	Reserved		
0x00_0204 0x00_0244 0x00_0284	(Read) UART Status Registers (USRn) (Write) UART Clock Select register <sup>1</sup> (UCSRn)	Reserved		
0x00_0208 0x00_0248 0x00_0288	(Read) Do not access <sup>2</sup> (Write) UART Command Registers (UCRn)	Reserved		
0x00_020C 0x00_024C 0x00_028C	(UART/Read) UART Receive Buffers (URBn) (UART/Write) UART Transmit Buffers (UTBn)	Reserved		

**Table 27-2. UART Module Memory Map (Continued)**

IPSBAR Offset				
UART0 UART1 UART2	[31:24]	[23:16]	[15:8]	[7:0]
0x00_0210 0x00_0250 0x00_0290	(Read) UART Input Port Change Register (UIPCR $n$ )	Reserved		
	(Write) UART Auxiliary Control Register <sup>1</sup> (UACR $n$ )	Reserved		
0x00_0214 0x00_0254 0x00_0294	(Read) UART interrupt Status Register (UISR $n$ )	Reserved		
	(Write) UART Interrupt Mask Register (UIMR $n$ )	Reserved		
0x00_0218 0x00_0258 0x00_0298	(Read) Do not access <sup>2</sup>	Reserved		
	(Write) UART Divider Upper Register (UBG1 $n$ )	Reserved		
0x00_021C 0x00_025C 0x00_029C	(Read) Do not access <sup>2</sup>	Reserved		
	(Write) UART Divider Lower Register (UBG2 $n$ )	Reserved		
0x00_0234 0x00_0274 0x00_02B4	(Read) UART Input Port Register (UIP $n$ )	Reserved		
	(Write) Do not access <sup>2</sup>	Reserved		
0x00_0238 0x00_0278 0x00_02B8	(Read) Do not access <sup>2</sup>	Reserved		
	(Write) UART Output Port Bit Set Command Register (UOP1 $n$ )	Reserved		
0x00_023C 0x00_027C 0x00_02BC	(Read) Do not access <sup>2</sup>	Reserved		
	(Write) UART Output Port Bit Reset Command Register (UOP0 $n$ )	Reserved		

<sup>1</sup> UMR1 $n$ , UMR2 $n$ , and UCSR $n$  should be changed only after the receiver/transmitter is issued a software reset command. That is, if channel operation is not disabled, undesirable results may occur.

<sup>2</sup> This address is for factory testing. Reading this location results in undesired effects and possible incorrect transmission or reception of characters. Register contents may also be changed.

### 27.3.1 UART Mode Registers 1 (UMR1 $n$ )

The UMR1 $n$  registers control configuration. UMR1 $n$  can be read or written when the mode register pointer points to it, at RESET or after a RESET MODE REGISTER POINTER command using UCR $n$ [MISC]. After UMR1 $n$  is read or written, the pointer points to UMR2 $n$ .

	7	6	5	4	3	2	1	0
R	RXRTS	RXIRQ/ FFULL	ERR	PM		PT	B/C	
W								
Reset	0	0	0	0	0	0	0	0
Address	IPSBAR + 0x0200 (UART0); IPSBAR + 0x0240 (UART1); IPSBAR + 0x0280 (UART2) After UMR1n is read or written, the pointer points to UMR2n.							

Figure 27-3. UART Mode Registers 1 (UMR1n)

Table 27-3. UMR1n Field Descriptions

Bits	Name	Description																				
7	RxRTS	Receiver request-to-send. Allows the $\overline{UnRTS}$ output to control the $\overline{UnCTS}$ input of the transmitting device to prevent receiver overrun. If both the receiver and transmitter are incorrectly programmed for $\overline{UnRTS}$ control, $\overline{UnRTS}$ control is disabled for both. Transmitter RTS control is configured in $UMR2n[TxRTS]$ . 0 The receiver has no effect on $\overline{UnRTS}$ . 1 When a valid start bit is received, $\overline{UnRTS}$ is negated if the UART's FIFO is full. $\overline{UnRTS}$ is reasserted when the FIFO has an empty position available.																				
6	RxIRQ/ FFULL	Receiver interrupt select. 0 RxRDY is the source that generates interrupt or DMA requests. 1 FFULL is the source that generates interrupt or DMA requests.																				
5	ERR	Error mode. Configures the FIFO status bits, $USRn[RB,FE,PE]$ . 0 Character mode. The $USRn$ values reflect the status of the character at the top of the FIFO. ERR must be 0 for correct A/D flag information when in multidrop mode. 1 Block mode. The $USRn$ values are the logical OR of the status for all characters reaching the top of the FIFO since the last RESET ERROR STATUS command for the channel was issued. See <a href="#">Section 27.3.5, "UART Command Registers (UCRn)."</a>																				
4–3	PM	Parity mode. Selects the parity or multidrop mode for the channel. The parity bit is added to the transmitted character, and the receiver performs a parity check on incoming data. The value of PM affects PT, as shown below.																				
2	PT	Parity type. PM and PT together select parity type (PM = 0x) or determine whether a data or address character is transmitted (PM = 11). <table border="1"><thead><tr><th>PM</th><th>Parity Mode</th><th>Parity Type (PT= 0)</th><th>Parity Type (PT= 1)</th></tr></thead><tbody><tr><td>00</td><td>With parity</td><td>Even parity</td><td>Odd parity</td></tr><tr><td>01</td><td>Force parity</td><td>Low parity</td><td>High parity</td></tr><tr><td>10</td><td>No parity</td><td colspan="2">n/a</td></tr><tr><td>11</td><td>Multidrop mode</td><td>Data character</td><td>Address character</td></tr></tbody></table>	PM	Parity Mode	Parity Type (PT= 0)	Parity Type (PT= 1)	00	With parity	Even parity	Odd parity	01	Force parity	Low parity	High parity	10	No parity	n/a		11	Multidrop mode	Data character	Address character
PM	Parity Mode	Parity Type (PT= 0)	Parity Type (PT= 1)																			
00	With parity	Even parity	Odd parity																			
01	Force parity	Low parity	High parity																			
10	No parity	n/a																				
11	Multidrop mode	Data character	Address character																			
1–0	B/C	Bits per character. Select the number of data bits per character to be sent. The values shown do not include start, parity, or stop bits. 00 5 bits 01 6 bits 10 7 bits 11 8 bits																				

## 27.3.2 UART Mode Register 2 (UMR2 $n$ )

The UMR2 $n$  registers control UART module configuration. UMR2 $n$  can be read or written when the mode register pointer points to it, which occurs after any access to UMR1 $n$ . UMR2 $n$  accesses do not update the pointer.

	7	6	5	4	3	2	1	0
R	CM		TXRTS	TXCTS	SB			
W								
Reset	0	0	0	0	0	0	0	0
Address	IPSBAR + 0x0200 (UART0); IPSBAR + 0x0240 (UART1); IPSBAR + 0x0280 (UART2) After UMR1 $n$ is read or written, the pointer points to UMR2 $n$ .							

**Figure 27-4. UART Mode Register 2 (UMR2 $n$ )**

**Table 27-4. UMR2 $n$  Field Descriptions**

Bits	Name	Description
7–6	CM	Channel mode. Selects a channel mode. <a href="#">Section 27.4.3, “Looping Modes,”</a> describes individual modes. 00 Normal 01 Automatic echo 10 Local loop-back 11 Remote loop-back
5	TxRTS	Transmitter ready-to-send. Controls negation of $\overline{UnRTS}$ to automatically terminate a message transmission. Attempting to program a receiver and transmitter in the same channel for $\overline{UnRTS}$ control is not permitted and disables $\overline{UnRTS}$ control for both. 0 The transmitter has no effect on $\overline{UnRTS}$ . 1 In applications where the transmitter is disabled after transmission completes, setting this bit automatically clears UOP[RTS] one bit time after any characters in the channel transmitter shift and holding registers are completely sent, including the programmed number of stop bits.

Table 27-4. UMR2n Field Descriptions (Continued)

Bits	Name	Description																																													
4	TxCTS	Transmitter clear-to-send. If both TxCTS and TxRTS are enabled, TxCTS controls the operation of the transmitter. 0 $\overline{UnCTS}$ has no effect on the transmitter. 1 Enables clear-to-send operation. The transmitter checks the state of $\overline{UnCTS}$ each time it is ready to send a character. If $\overline{UnCTS}$ is asserted, the character is sent; if it is deasserted, the channel $\overline{UnTXD}$ remains in the high state and transmission is delayed until $\overline{UnCTS}$ is asserted. Changes in $\overline{UnCTS}$ as a character is being sent do not affect its transmission.																																													
3–0	SB	Stop-bit length control. Selects the length of the stop bit appended to the transmitted character. Stop-bit lengths of 9/16 to 2 bits are programmable for 6–8 bit characters. Lengths of 1-1/16 to 2 bits are programmable for 5-bit characters. In all cases, the receiver checks only for a high condition at the center of the first stop-bit position, that is, one bit time after the last data bit or after the parity bit, if parity is enabled. If an external 1x clock is used for the transmitter, clearing bit 3 selects one stop bit and setting bit 3 selects two stop bits for transmission. <table><tr><th>SB</th><th>5 Bits</th><th>6–8 Bits</th><th>SB</th><th>5–8 Bits</th></tr><tr><td>0000</td><td>1.063</td><td>0.563</td><td>1000</td><td>1.563</td></tr><tr><td>0001</td><td>1.125</td><td>0.625</td><td>1001</td><td>1.625</td></tr><tr><td>0010</td><td>1.188</td><td>0.688</td><td>1010</td><td>1.688</td></tr><tr><td>0011</td><td>1.250</td><td>0.750</td><td>1011</td><td>1.750</td></tr><tr><td>0100</td><td>1.313</td><td>0.813</td><td>1100</td><td>1.813</td></tr><tr><td>0101</td><td>1.375</td><td>0.875</td><td>1101</td><td>1.875</td></tr><tr><td>0110</td><td>1.438</td><td>0.938</td><td>1110</td><td>1.938</td></tr><tr><td>0111</td><td>1.500</td><td>1.000</td><td>1111</td><td>2.000</td></tr></table>	SB	5 Bits	6–8 Bits	SB	5–8 Bits	0000	1.063	0.563	1000	1.563	0001	1.125	0.625	1001	1.625	0010	1.188	0.688	1010	1.688	0011	1.250	0.750	1011	1.750	0100	1.313	0.813	1100	1.813	0101	1.375	0.875	1101	1.875	0110	1.438	0.938	1110	1.938	0111	1.500	1.000	1111	2.000
SB	5 Bits	6–8 Bits	SB	5–8 Bits																																											
0000	1.063	0.563	1000	1.563																																											
0001	1.125	0.625	1001	1.625																																											
0010	1.188	0.688	1010	1.688																																											
0011	1.250	0.750	1011	1.750																																											
0100	1.313	0.813	1100	1.813																																											
0101	1.375	0.875	1101	1.875																																											
0110	1.438	0.938	1110	1.938																																											
0111	1.500	1.000	1111	2.000																																											

### 27.3.3 UART Status Registers (USRn)

The USRn registers, shown in Figure 27-5, show the status of the transmitter, the receiver, and the FIFO.

	7	6	5	4	3	2	1	0
R	RB	FE	PE	OE	TXEMP	TXRDY	FFULL	RXRDY
W								
Reset	0	0	0	0	0	0	0	0
Address	IPSBAR + 0x0204 (USR0); IPSBAR + 0x0244 (USR1); IPSBAR + 0x0284 (USR2)							

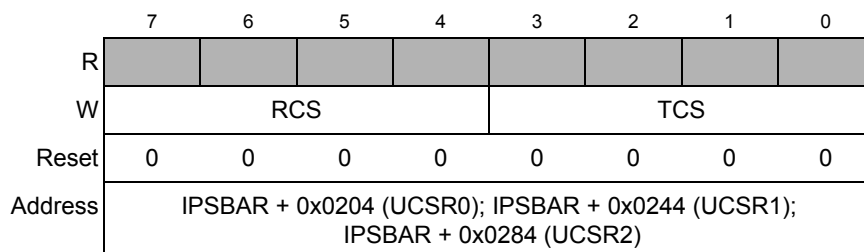
Figure 27-5. UART Status Register (USRn)

**Table 27-5. USR<sub>n</sub> Field Descriptions**

Bits	Name	Description
7	RB	Received break. The received break circuit detects breaks that originate in the middle of a received character. However, a break in the middle of a character must persist until the end of the next detected character time. 0 No break was received. 1 An all-zero character of the programmed length was received without a stop bit. Only a single FIFO position is occupied when a break is received. Further entries to the FIFO are inhibited until U <sub>n</sub> RXD returns to the high state for at least one-half bit time, which is equal to two successive edges of the UART clock. RB is valid only when RxRDY = 1.
6	FE	Framing error. 0 No framing error occurred. 1 No stop bit was detected when the corresponding data character in the FIFO was received. The stop-bit check occurs in the middle of the first stop-bit position. FE is valid only when RxRDY = 1.
5	PE	Parity error. Valid only if RxRDY = 1. 0 No parity error occurred. 1 If UMR1 <sub>n</sub> [PM] = 0x (with parity or force parity), the corresponding character in the FIFO was received with incorrect parity. If UMR1 <sub>n</sub> [PM] = 11 (multidrop), PE stores the received address or data (A/D) bit. PE is valid only when RxRDY = 1.
4	OE	Overrun error. Indicates whether an overrun occurs. 0 No overrun occurred. 1 One or more characters in the received data stream have been lost. OE is set upon receipt of a new character when the FIFO is full and a character is already in the shift register waiting for an empty FIFO position. When this occurs, the character in the receiver shift register and its break detect, framing error status, and parity error, if any, are lost. OE is cleared by the RESET ERROR STATUS command in UCR <sub>n</sub> .
3	TxEMP	Transmitter empty. 0 The transmit buffer is not empty. Either a character is being shifted out, or the transmitter is disabled. The transmitter is enabled/disabled by programming UCR <sub>n</sub> [TC]. 1 The transmitter has underrun (both the transmitter holding register and transmitter shift registers are empty). This bit is set after transmission of the last stop bit of a character if there are no characters in the transmitter holding register awaiting transmission.
2	TxRDY	Transmitter ready. 0 The CPU loaded the transmitter holding register or the transmitter is disabled. 1 The transmitter holding register is empty and ready for a character. TxRDY is set when a character is sent to the transmitter shift register or when the transmitter is first enabled. If the transmitter is disabled, characters loaded into the transmitter holding register are not sent.
1	FFULL	FIFO full. 0 The FIFO is not full but may hold up to two unread characters. 1 A character was received and the receiver FIFO is now full. Any characters received when the FIFO is full are lost.
0	RxRDY	Receiver ready. 0 The CPU has read the receive buffer and no characters remain in the FIFO after this read. 1 One or more characters were received and are waiting in the receive buffer FIFO.

## 27.3.4 UART Clock Select Registers (UCSR<sub>n</sub>)

The UCSRs select an external clock on the DTIN input (divided by 1 or 16) or a prescaled internal bus clock as the clocking source for the transmitter and receiver. See [Section 27.4.1, “Transmitter/Receiver Clock Source.”](#) The transmitter and receiver can use different clock sources. To use the internal bus clock for both, set UCSR<sub>n</sub> to 0xDD.



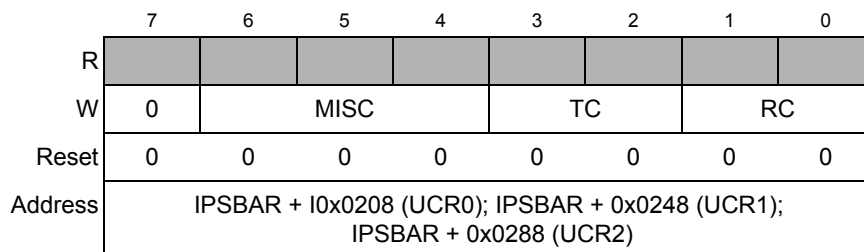
**Figure 27-6. UART Clock Select Register (UCSR<sub>n</sub>)**

**Table 27-6. UCSR<sub>n</sub> Field Descriptions**

Bits	Name	Description
7–4	RCS	Receiver clock select. Selects the clock source for the receiver channel. 1101 Prescaled internal bus clock 1110 DTIN divided by 16 1111 DTIN
3–0	TCS	Transmitter clock select. Selects the clock source for the transmitter channel. 1101 Prescaled internal bus clock 1110 DTIN divided by 16 1111 DTIN

## 27.3.5 UART Command Registers (UCR<sub>n</sub>)

The UCRs, shown in [Figure 27-7](#), supply commands to the UART. Only multiple commands that do not conflict can be specified in a single write to a UCR<sub>n</sub>. For example, RESET TRANSMITTER and ENABLE TRANSMITTER cannot be specified in one command.



**Figure 27-7. UART Command Register (UCR<sub>n</sub>)**

[Table 27-7](#) describes UCR<sub>n</sub> fields and commands. Examples in [Section 27.4.2, “Transmitter and Receiver Operating Modes,”](#) show how these commands are used.



Table 27-7. UCR<sub>n</sub> Field Descriptions

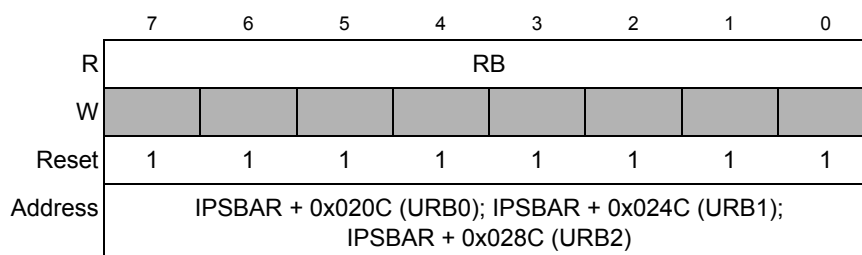
Bits	Value	Command	Description
6–4	<b>MISC Field</b> (This field selects a single command.)		
	000	NO COMMAND	—
	001	RESET MODE REGISTER POINTER	Causes the mode register pointer to point to UMR1 <sub>n</sub> .
	010	RESET RECEIVER	Immediately disables the receiver, clears USR <sub>n</sub> [FFULL,RxRDY], and reinitializes the receiver FIFO pointer. No other registers are altered. Because it places the receiver in a known state, use this command instead of RECEIVER DISABLE when reconfiguring the receiver.
	011	RESET TRANSMITTER	Immediately disables the transmitter and clears USR <sub>n</sub> [TxEMP,TxRDY]. No other registers are altered. Because it places the transmitter in a known state, use this command instead of TRANSMITTER DISABLE when reconfiguring the transmitter.
	100	RESET ERROR STATUS	Clears USR <sub>n</sub> [RB,FE,PE,OE]. Also used in block mode to clear all error bits after a data block is received.
	101	RESET BREAK—CHANGE INTERRUPT	Clears the delta break bit, UISR <sub>n</sub> [DB].
	110	START BREAK	Forces UnTXD low. If the transmitter is empty, the break may be delayed up to one bit time. If the transmitter is active, the break starts when character transmission completes. The break is delayed until any character in the transmitter shift register is sent. Any character in the transmitter holding register is sent after the break. The transmitter must be enabled for the command to be accepted. This command ignores the state of UnCTS.
	111	STOP BREAK	Causes UnTXD to go high (mark) within two bit times. Any characters in the transmit buffer are sent.
3–2	<b>TC Field</b> (This field selects a single command)		
	00	NO ACTION TAKEN	Causes the transmitter to stay in its current mode: if the transmitter is enabled, it remains enabled; if the transmitter is disabled, it remains disabled.
	01	TRANSMITTER ENABLE	Enables operation of the channel's transmitter. USR <sub>n</sub> [TxEMP,TxRDY] are set. If the transmitter is already enabled, this command has no effect.
	10	TRANSMITTER DISABLE	Terminates transmitter operation and clears USR <sub>n</sub> [TxEMP,TxRDY]. If a character is being sent when the transmitter is disabled, transmission completes before the transmitter becomes inactive. If the transmitter is already disabled, the command has no effect.
	11	—	Reserved, do not use.

**Table 27-7. UCR<sub>n</sub> Field Descriptions (Continued)**

Bits	Value	Command	Description
1–0	RC (This field selects a single command)		
	00	NO ACTION TAKEN	Causes the receiver to stay in its current mode. If the receiver is enabled, it remains enabled; if disabled, it remains disabled.
	01	RECEIVER ENABLE	If the UART module is not in multidrop mode (UMR1 <sub>n</sub> [PM] ≠ 11), RECEIVER ENABLE enables the channel's receiver and forces it into search-for-start-bit state. If the receiver is already enabled, this command has no effect.
	10	RECEIVER DISABLE	Disables the receiver immediately. Any character being received is lost. The command does not affect receiver status bits or other control registers. If the UART module is programmed for local loop-back or multidrop mode, the receiver operates even though this command is selected. If the receiver is already disabled, the command has no effect.
	11	—	Reserved, do not use.

### 27.3.6 UART Receive Buffers (URB<sub>n</sub>)

The receive buffers (shown in [Figure 27-8](#)) contain one serial shift register and three receiver holding registers, which act as a FIFO. UnRXD is connected to the serial shift register. The CPU reads from the top of the FIFO while the receiver shifts and updates from the bottom when the shift register is full (see [Figure 27-18](#)). RB contains the character in the receiver.

**Figure 27-8. UART Receive Buffer (URB<sub>n</sub>)**

### 27.3.7 UART Transmit Buffers (UTB<sub>n</sub>)

The transmit buffers consist of the transmitter holding register and the transmitter shift register. The holding register accepts characters from the bus master if channel's USR<sub>n</sub>[TxRDY] is set. A write to the transmit buffer clears USR<sub>n</sub>[TxRDY], inhibiting any more characters until the shift register can accept more data. When the shift register is empty, it checks if the holding register has a valid character to be sent (TxRDY = 0). If there is a valid character, the shift register loads it and sets USR<sub>n</sub>[TxRDY] again. Writes to the transmit buffer when the channel's TxRDY = 0 and when the transmitter is disabled have no effect on the transmit buffer.

[Figure 27-9](#) shows UTB<sub>n</sub>. TB contains the character in the transmit buffer.

	7	6	5	4	3	2	1	0
R								
W	TB							
Reset	0	0	0	0	0	0	0	0
Address	IPSBAR + 0x020C(UTB0); IPSBAR + 0x024C(UTB1); IPSBAR + 0x028C(UTB2)							

Figure 27-9. UART Transmit Buffer (UTB<sub>n</sub>)

### 27.3.8 UART Input Port Change Registers (UIPCR<sub>n</sub>)

The UIPCRs, shown in Figure 27-10, hold the current state and the change-of-state for  $\overline{UnCTS}$ .

	7	6	5	4	3	2	1	0
R	0	0	0	COS	0	0	0	CTS
W								
Reset	0	0	0	0	1	1	1	$\overline{UnCTS}$
Address	IPSBAR + 0x0210 (UIPCR0); IPSBAR + 0x0250 (UIPCR1); IPSBAR + 0x0290 (UIPCR2)							

Figure 27-10. UART Input Port Change Register (UIPCR<sub>n</sub>)Table 27-8. UIPCR<sub>n</sub> Field Descriptions

Bits	Name	Description
7–5	—	Reserved, should be cleared.
4	COS	Change of state (high-to-low or low-to-high transition). 0 No change-of-state since the CPU last read UIPCR <sub>n</sub> . Reading UIPCR <sub>n</sub> clears UISR <sub>n</sub> [COS]. 1 A change-of-state longer than 25–50 μs occurred on the $\overline{UnCTS}$ input. UACR <sub>n</sub> can be programmed to generate an interrupt to the CPU when a change of state is detected.
3–1	—	Reserved, should be cleared.
0	CTS	Current state of clear-to-send. Starting two serial clock periods after reset, CTS reflects the state of $\overline{UnCTS}$ . If $\overline{UnCTS}$ is detected asserted at that time, COS is set, which initiates an interrupt if UACR <sub>n</sub> [IEC] is enabled. 0 The current state of the $\overline{UnCTS}$ input is asserted. 1 The current state of the $\overline{UnCTS}$ input is deasserted.

### 27.3.9 UART Auxiliary Control Register (UACR<sub>n</sub>)

The UACRs, shown in Figure 27-8, control the input enable.

	7	6	5	4	3	2	1	0
R								
W	0	0	0	0	0	0	0	IEC
Reset	0	0	0	0	0	0	0	0
Address	IPSBAR + 0x0210 (UACR0); IPSBAR + 0x0250 (UACR1); IPSBAR + 0x0290 (UACR2)							

Figure 27-11. UART Auxiliary Control Register (UACR<sub>n</sub>)Table 27-9. UACR<sub>n</sub> Field Descriptions

Bits	Name	Description
7–1	—	Reserved, should be cleared.
0	IEC	Input enable control. 0 Setting the corresponding UIPCR <sub>n</sub> bit has no effect on UISR <sub>n</sub> [COS]. 1 UISR <sub>n</sub> [COS] is set and an interrupt is generated when the UIPCR <sub>n</sub> [COS] is set by an external transition on the UnCTS input (if UIMR <sub>n</sub> [COS] = 1).

### 27.3.10 UART Interrupt Status/Mask Registers (UISR<sub>n</sub>/UIMR<sub>n</sub>)

The UISRs, shown in Figure 27-12, provide status for all potential interrupt sources. UISR<sub>n</sub> contents are masked by UIMR<sub>n</sub>. If corresponding UISR<sub>n</sub> and UIMR<sub>n</sub> bits are set, the internal interrupt output is asserted. If a UIMR<sub>n</sub> bit is cleared, the state of the corresponding UISR<sub>n</sub> bit has no effect on the output.

The UISR<sub>n</sub> and UIMR<sub>n</sub> registers share the same space in memory. Reading this register provides the user with interrupt status, while writing controls the mask bits.

#### NOTE

True status is provided in the UISR<sub>n</sub> regardless of UIMR<sub>n</sub> settings.  
UISR<sub>n</sub> is cleared when the UART module is reset.

	7	6	5	4	3	2	1	0
R (UISR <sub>n</sub> )	COS	0	0	0	0	DB	FFULL/ RXRDY	TXRDY
W (UIMR <sub>n</sub> )	COS	0	0	0	0	DB	FFULL/ RXRDY	TXRDY
Reset	0	0	0	0	0	0	0	0
Address	IPSBAR + 0x0214 (UISR0); IPSBAR + 0x0254 (UISR1); IPSBAR + 0x0294 (UISR2)							

Figure 27-12. UART Interrupt Status/Mask Registers (UISR<sub>n</sub>/UIMR<sub>n</sub>)

Table 27-10. UISR $n$ /UIMR $n$  Field Descriptions

Bits	Name	Description																						
7	COS	Change-of-state. 0 UIPCRn[COS] is not selected. 1 Change-of-state occurred on UnCTS and was programmed in UACRn[IEC] to cause an interrupt.																						
6–3	—	Reserved, should be cleared.																						
2	DB	Delta break. 0 No new break-change condition to report. <a href="#">Section 27.3.5, “UART Command Registers (UCRn),”</a> describes the RESET BREAK-CHANGE INTERRUPT command. 1 The receiver detected the beginning or end of a received break.																						
1	FFULL/ RxRDY	Status of FIFO or receiver, depending on UMR1[FFULL/RxRDY] bit. Duplicate of USRn[FIFO] & USRn[RxRDY] <table><tr><th rowspan="2">UIMRn [FFULL/RxRDY ]</th><th rowspan="2">UISRn [FFULL/RxRDY]</th><th colspan="2">UMR1n[FFULL/RxRDY]</th></tr><tr><th>0 (RxRDY)</th><th>1 (FIFO)</th></tr><tr><td>0</td><td>0</td><td>Receiver not ready</td><td>FIFO not full</td></tr><tr><td>1</td><td>0</td><td>Receiver not ready</td><td>FIFO not full</td></tr><tr><td>0</td><td>1</td><td>Receiver is ready, Do not interrupt</td><td>FIFO is full, Do not interrupt</td></tr><tr><td>1</td><td>1</td><td>Receiver is ready, interrupt</td><td>FIFO is full, interrupt</td></tr></table>	UIMRn [FFULL/RxRDY ]	UISRn [FFULL/RxRDY]	UMR1n[FFULL/RxRDY]		0 (RxRDY)	1 (FIFO)	0	0	Receiver not ready	FIFO not full	1	0	Receiver not ready	FIFO not full	0	1	Receiver is ready, Do not interrupt	FIFO is full, Do not interrupt	1	1	Receiver is ready, interrupt	FIFO is full, interrupt
UIMRn [FFULL/RxRDY ]	UISRn [FFULL/RxRDY]	UMR1n[FFULL/RxRDY]																						
		0 (RxRDY)	1 (FIFO)																					
0	0	Receiver not ready	FIFO not full																					
1	0	Receiver not ready	FIFO not full																					
0	1	Receiver is ready, Do not interrupt	FIFO is full, Do not interrupt																					
1	1	Receiver is ready, interrupt	FIFO is full, interrupt																					
0	TxRDY	Transmitter ready. This bit is the duplication of USRn[TxRDY]. 0 The transmitter holding register was loaded by the CPU or the transmitter is disabled. Characters loaded into the transmitter holding register when TxRDY = 0 are not sent. 1 The transmitter holding register is empty and ready to be loaded with a character.																						

### 27.3.11 UART Baud Rate Generator Registers (UBG1 $n$ /UBG2 $n$ )

The UBG1 $n$  registers hold the msb, and the UBG2 $n$  registers hold the lsb of the preload value. UBG1 $n$  and UBG2 $n$  concatenate to provide a divider to the internal bus clock for transmitter/receiver operation, as described in [Section 27.4.1.2.1, “internal Bus Clock Baud Rates.”](#)

	7	6	5	4	3	2	1	0
R								
W	Divider MSB							
Reset	0	0	0	0	0	0	0	0
Address	IPSBAR + 0x0218 (UBG10); IPSBAR + 0x0258 (UBG11); IPSBAR + 0x0298 (UBG12)							

Figure 27-13. UART Baud Rate Generator Register (UBG1 $n$ )

	7	6	5	4	3	2	1	0
R								
W	Divider LSB							
Reset	0	0	0	0	0	0	0	0
Address	IPSBAR + 0x021C (UBG20); IPSBAR + 0x025C (UBG21); IPSBAR + 0x029C (UBG22)							

Figure 27-14. UART Baud Rate Generator Register (UBG2n)

**NOTE**

The minimum value that can be loaded on the concatenation of UBG1n with UBG2n is 0x0002. The UBG2n reset value of 0x00 is invalid and must be written to before the UART transmitter or receiver are enabled. Both UBG1n and UBG2n are write-only and cannot be read by the CPU.

**27.3.12 UART Input Port Register (UIPn)**

The UIPn registers, shown in Figure 27-15, show the current state of the  $\overline{UnCTS}$  input.

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	CTS
W								
Reset	1	1	1	1	1	1	1	1
Address	IPSBAR + 0x0234 (UIP0); IPSBAR + 0x0274 (UIP1); IPSBAR + 0x02B4 (UIP2)							

Figure 27-15. UART Input Port Register (UIPn)

**Table 27-11. UIPn Field Descriptions**

Bits	Name	Description
7–1	—	Reserved
0	CTS	Current state of clear-to-send. The $\overline{UnCTS}$ value is latched and reflects the state of the input pin when UIPn is read. Note: This bit has the same function and value as UIPCRn[RTS]. 0 The current state of the $\overline{UnCTS}$ input is logic 0. 1 The current state of the $\overline{UnCTS}$ input is logic 1.

**27.3.13 UART Output Port Command Registers (UOP1n/UOP0n)**

The  $\overline{UnRTS}$  output can be asserted by writing a 1 to UOP1n[RTS] and negated by writing a 1 to UOP0n[RTS]. See Figure 27-16.

	7	6	5	4	3	2	1	0
R								
W	0	0	0	0	0	0	0	RTS
Reset	0	0	0	0	0	0	0	0
Address	UART0: IPSBAR + 0x0238 (UOP1), IPSBAR + 0x023C (UOP0) UART1: IPSBAR + 0x0278 (UOP1), IPSBAR + 0x027C (UOP0) UART2: IPSBAR + 0x02B8 (UOP1) IPSBAR + 0x02BC (UOP0)							

Figure 27-16. UART Output Port Command Registers (UOP1n/UOP0n)

Table 27-12. UOP1/UOP0 Field Descriptions

Bits	Name	Description
7–1	—	Reserved, should be cleared.
0	RTS	Output port output. Controls assertion (UOP1)/negation (UOP0) of $\overline{U_nRTS}$ output. 0 Not affected. 1 Asserts $\overline{U_nRTS}$ in UOP1. Negates $\overline{U_nRTS}$ in UOP0.

## 27.4 Functional Description

This section describes operation of the clock source generator, transmitter, and receiver.

### 27.4.1 Transmitter/Receiver Clock Source

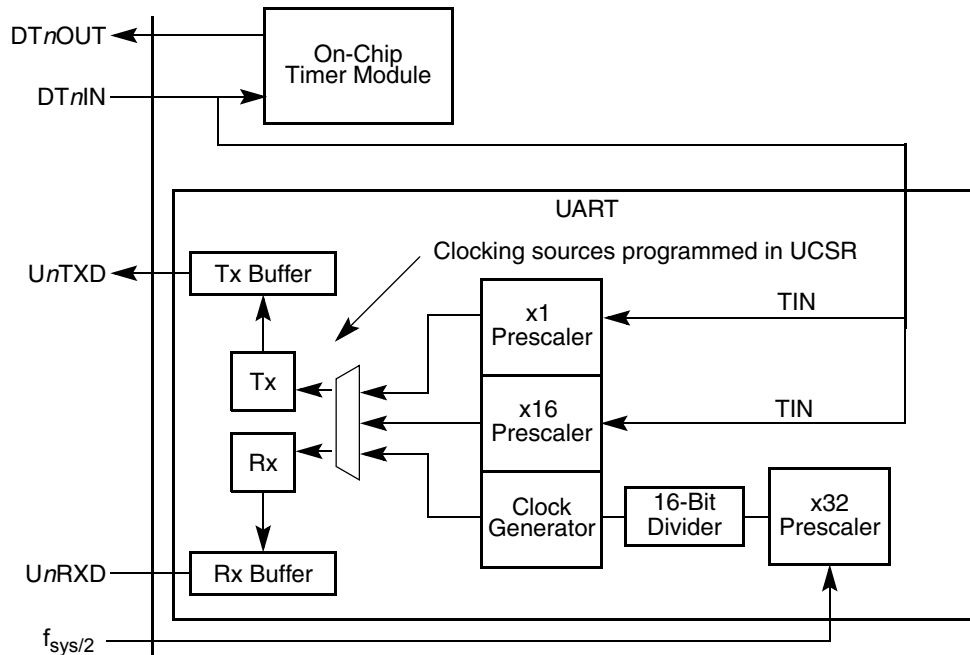
The internal bus clock serves as the basic timing reference for the clock source generator logic, which consists of a clock generator and a programmable 16-bit divider dedicated to each UART. The clock generator might not produce standard baud rates if the internal bus clock is used, so enable the 16-bit divider.

#### 27.4.1.1 Programmable Divider

As [Figure 27-17](#) shows, the UART $n$  transmitter and receiver can use the following clock sources:

- An external clock signal on the DT $n$ IN pin. When not divided, DT $n$ IN provides a synchronous clock mode; when divided by 16, it is asynchronous.
- The internal bus clock supplies an asynchronous clock source that is divided by 32 and then divided by the 16-bit value programmed in UBG1 $n$  and UBG2 $n$ . See [Section 27.3.11](#), “UART Baud Rate Generator Registers (UBG1n/UBG2n).”

The choice of DTIN or internal bus clock is programmed in the UCSR.



**Figure 27-17. Clocking Source Diagram**

### NOTE

If DTnIN is a clocking source for either the timer or UART, that timer module cannot use DTnIN for timer input capture.

## 27.4.1.2 Calculating Baud Rates

The following sections describe how to calculate baud rates.

### 27.4.1.2.1 internal Bus Clock Baud Rates

When the internal bus clock is the UART clocking source, it goes through a divide-by-32 prescaler and then passes through the 16-bit divider of the concatenated UBG1n and UBG2n registers. The baud-rate calculation is as follows:

$$\text{Baudrate} = \frac{f_{\text{sys}/2}}{[32 \times \text{divider}]}$$

Using a 83 MHz internal bus clock and letting baud rate = 9600, then

$$\text{Divider} = \frac{83\text{MHz}}{[32 \times 9600]} = 270(\text{decimal}) = 0x0103(\text{hexadecimal})$$

therefore UBG1n = 0x01 and UBG2n = 0xF403.



### 27.4.1.2.2 External Clock

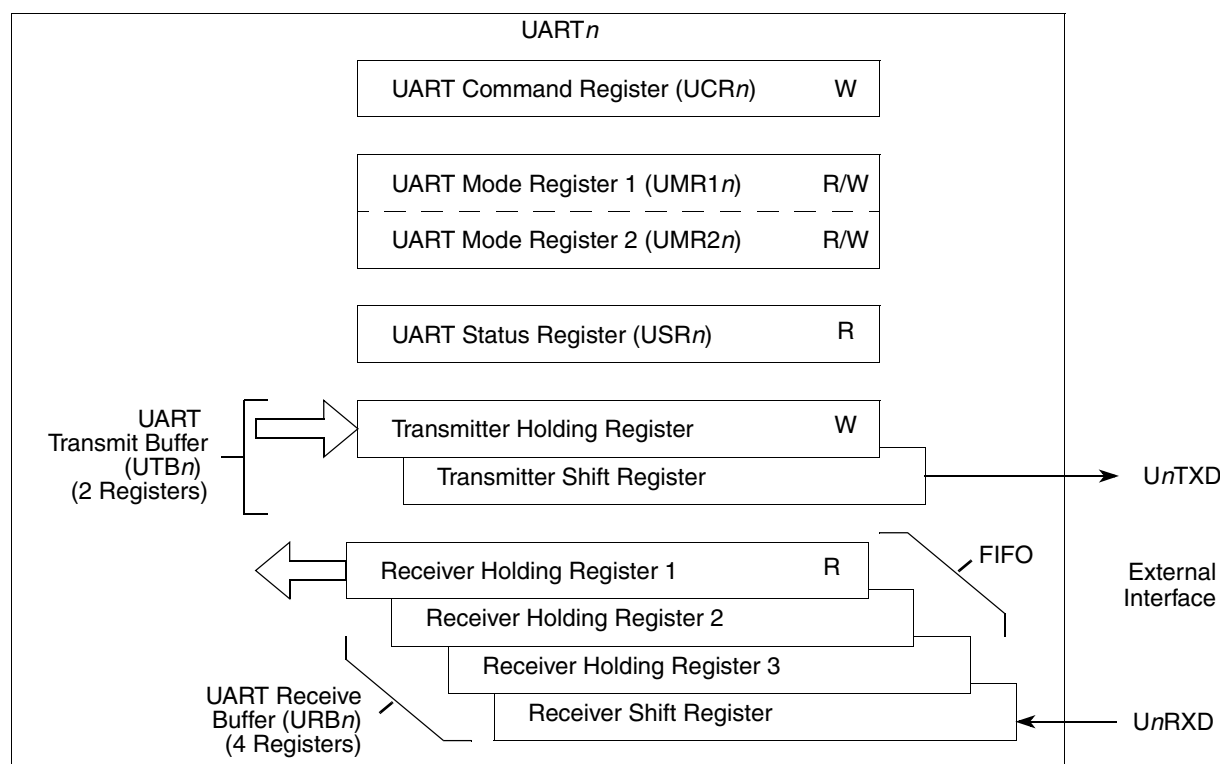
An external source clock (DT $n$ IN) can be used as is or divided by 16.

If  $f_{\text{extc}}$  is the external clock frequency, then the baud rate can be described with this equation:

$$\text{Baudrate} = \frac{f_{\text{extc}}}{(16 \text{ or } 1)}$$

## 27.4.2 Transmitter and Receiver Operating Modes

Figure 27-18 is a functional block diagram of the transmitter and receiver showing the command and operating registers, which are described generally in the following sections and described in detail in Section 27.3, “Memory Map/Register Definition.”



**Figure 27-18. Transmitter and Receiver Functional Diagram**

### 27.4.2.1 Transmitter

The transmitter is enabled through the UART command register (UCR $n$ ). When it is ready to accept a character, the UART sets USR $n$ [TxRDY]. The transmitter converts parallel data from the CPU to a serial bit stream on UnTXD. It automatically sends a start bit followed by the programmed number of data bits, an optional parity bit, and the programmed number of stop bits. The lsb is sent first. Data is shifted from the transmitter output on the falling edge of the clock source.

After the stop bits are sent, if no new character is in the transmitter holding register, the  $UnTXD$  output remains high (mark condition) and the transmitter empty bit,  $USRn[TxEMP]$ , is set. Transmission resumes and  $TxEMP$  is cleared when the CPU loads a new character into the UART transmit buffer ( $UTBn$ ). If the transmitter receives a disable command, it continues until any character in the transmitter shift register is completely sent.

If the transmitter is reset through a software command, operation stops immediately (see [Section 27.3.5, “UART Command Registers \(UCRn\)”](#)). The transmitter is reenabled through the  $UCRn$  to resume operation after a disable or software reset.

If the clear-to-send operation is enabled,  $\overline{UnCTS}$  must be asserted for the character to be transmitted. If  $\overline{UnCTS}$  is negated in the middle of a transmission, the character in the shift register is sent and  $UnTXD$  remains in mark state until  $\overline{UnCTS}n$  is reasserted. If the transmitter is forced to send a continuous low condition by issuing a SEND BREAK command, the transmitter ignores the state of  $\overline{UnCTS}$ .

If the transmitter is programmed to automatically negate  $\overline{UnRTS}$  when a message transmission completes,  $\overline{UnRTS}$  must be asserted manually before a message is sent. In applications in which the transmitter is disabled after transmission is complete and  $\overline{UnRTS}$  is appropriately programmed,  $\overline{UnRTS}$  is negated one bit time after the character in the shift register is completely transmitted. The transmitter must be manually reenabled by reasserting  $\overline{UnRTS}$  before the next message is to be sent.

[Figure 27-19](#) shows the functional timing information for the transmitter.

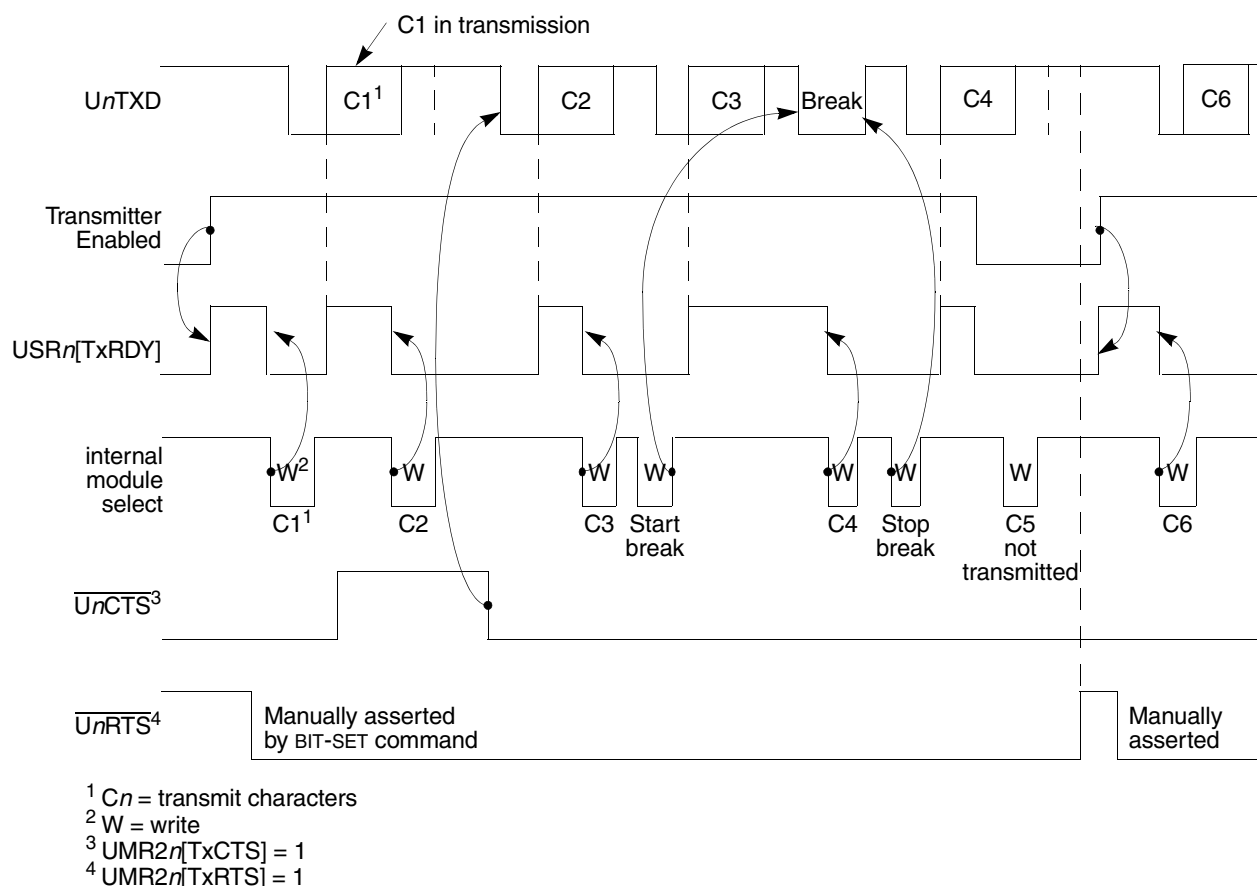


Figure 27-19. Transmitter Timing Diagram

### 27.4.2.2 Receiver

The receiver is enabled through its  $UCR_n$ , as described in [Section 27.3.5, “UART Command Registers \( \$UCR\_n\$ \)”](#).

When the receiver detects a high-to-low (mark-to-space) transition of the start bit on  $UnRXD$ , the state of  $UnRXD$  is sampled eight times on the edge of the bit time clock starting one-half clock after the transition (asynchronous operation) or at the next rising edge of the bit time clock (synchronous operation). If  $UnRXD$  is sampled high, the start bit is invalid and the search for the valid start bit begins again.

If  $UnRXD$  is still low, a valid start bit is assumed and the receiver continues sampling the input at one-bit time intervals, at the theoretical center of the bit, until the proper number of data bits and parity, if any, is assembled and one stop bit is detected. Data on the  $UnRXD$  input is sampled on the rising edge of the programmed clock source. The lsb is received first. The data is then transferred to a receiver holding register and  $USR_n[RxRDY]$  is set. If the character is less than eight bits, the most significant unused bits in the receiver holding register are cleared.

After the stop bit is detected, the receiver immediately looks for the next start bit. However, if a non-zero character is received without a stop bit (framing error) and  $UnRXD$  remains low for one-half of the bit period after the stop bit is sampled, the receiver operates as if a new start bit were detected. Parity error, framing error, overrun error, and received break conditions set the respective PE, FE, OE, RB error and break flags in the  $USRn$  at the received character boundary and are valid only if  $USRn[RxRDY]$  is set.

If a break condition is detected ( $UnRXD$  is low for the entire character including the stop bit), a character of all zeros is loaded into the receiver holding register and  $USRn[RB, RxRDY]$  are set.  $UnRXD$  must return to a high condition for at least one-half bit time before a search for the next start bit begins.

The receiver detects the beginning of a break in the middle of a character if the break persists through the next character time. If the break begins in the middle of a character, the receiver places the damaged character in the Rx FIFO and sets the corresponding  $USRn$  error bits and  $USRn[RxRDY]$ . Then, if the break lasts until the next character time, the receiver places an all-zero character into the Rx FIFO and sets  $USRn[RB, RxRDY]$ .

Figure 27-20 shows receiver functional timing.

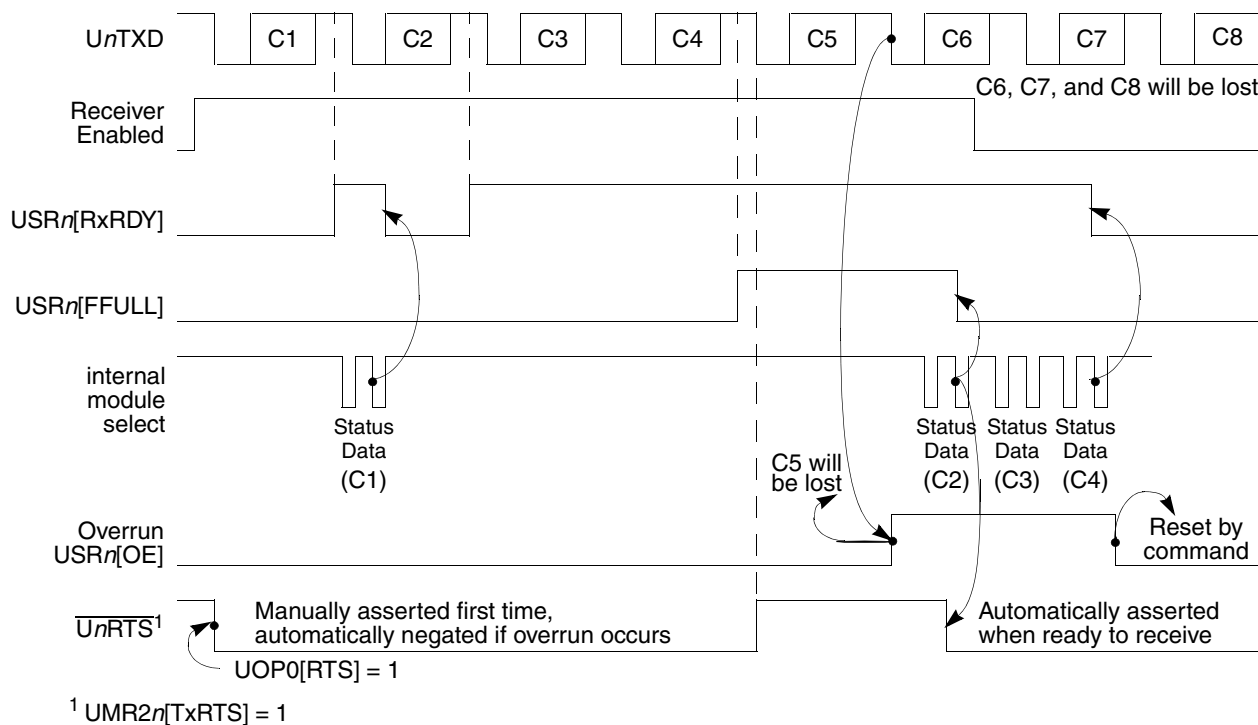


Figure 27-20. Receiver Timing

### 27.4.2.3 FIFO

The FIFO is used in the UART's receive buffer logic. The FIFO consists of three receiver holding registers. The receive buffer consists of the FIFO and a receiver shift register connected to the  $UnRXD$  (see Figure 27-18). Data is assembled in the receiver shift register and loaded into the top empty receiver holding register position of the FIFO. Thus, data flowing from the receiver to the CPU is quadruple-buffered.

In addition to the data byte, three status bits, parity error (PE), framing error (FE), and received break (RB), are appended to each data character in the FIFO; OE (overrun error) is not appended. By programming the ERR bit in the channel's mode register ( $UMR1n$ ), status is provided in character or block modes.

$USRn[RxRDY]$  is set when at least one character is available to be read by the CPU. A read of the receive buffer produces an output of data from the top of the FIFO. After the read cycle, the data at the top of the FIFO and its associated status bits are popped and the receiver shift register can add new data at the bottom of the FIFO. The FIFO-full status bit (FFULL) is set if all three positions are filled with data. Either the RxRDY or FFULL bit can be selected to cause an interrupt and either TxRDY or RxRDY can be used to generate a DMA request.

The two error modes are selected by  $UMR1n[ERR]$  as follows:

- In character mode ( $UMR1n[ERR] = 0$ ), status is given in the  $USRn$  for the character at the top of the FIFO.
- In block mode, the  $USRn$  shows a logical OR of all characters reaching the top of the FIFO since the last RESET ERROR STATUS command. Status is updated as characters reach the top of the FIFO. Block mode offers a data-reception speed advantage where the software overhead of error-checking each character cannot be tolerated. However, errors are not detected until the check is performed at the end of an entire message—the faulting character is not identified.

In either mode, reading the  $USRn$  does not affect the FIFO. The FIFO is popped only when the receive buffer is read. The  $USRn$  should be read before reading the receive buffer. If all three receiver holding registers are full, a new character is held in the receiver shift register until space is available. However, if a second new character is received, the contents of the character in the receiver shift register is lost, the FIFOs are unaffected, and  $USRn[OE]$  is set when the receiver detects the start bit of the new overrunning character.

To support flow control, the receiver can be programmed to automatically negate and assert  $\overline{UnRTS}$ , in which case the receiver automatically negates  $\overline{UnRTS}$  when a valid start bit is detected and the FIFO is full. The receiver asserts  $\overline{UnRTS}$  when a FIFO position becomes available; therefore, overrun errors can be prevented by connecting  $\overline{UnRTS}$  to the  $\overline{UnCTS}$  input of the transmitting device.

**NOTE**

The receiver can still read characters in the FIFO if the receiver is disabled. If the receiver is reset, the FIFO,  $\overline{UnRTS}$  control, all receiver status bits, and interrupts, and DMA requests are reset. No more characters are received until the receiver is reenabled.

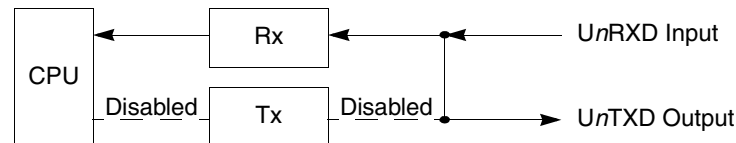
### 27.4.3 Looping Modes

The UART can be configured to operate in various looping modes as shown in [Figure 27-20](#). These modes are useful for local and remote system diagnostic functions. The modes are described in the following paragraphs and in [Section 27.3, “Memory Map/Register Definition.”](#)

The UART’s transmitter and receiver should be disabled when switching between modes. The selected mode is activated immediately upon mode selection, regardless of whether a character is being received or transmitted.

#### 27.4.3.1 Automatic Echo Mode

In automatic echo mode, shown in [Figure 27-21](#), the UART automatically resends received data bit by bit. The local CPU-to-receiver communication continues normally, but the CPU-to-transmitter link is disabled. In this mode, received data is clocked on the receiver clock and re-sent on  $UnTXD$ . The receiver must be enabled, but the transmitter need not be.

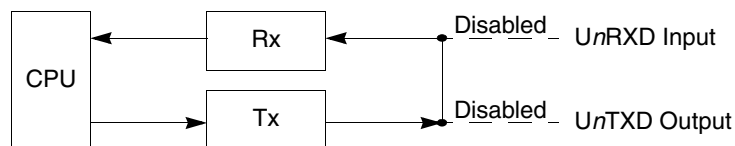


**Figure 27-21. Automatic Echo**

Because the transmitter is inactive,  $USR_n[TxEMP, TxRDY]$  are inactive and data is sent as it is received. Received parity is checked but is not recalculated for transmission. Character framing is also checked, but stop bits are sent as they are received. A received break is echoed as received until the next valid start bit is detected.

#### 27.4.3.2 Local Loop-Back Mode

[Figure 27-22](#) shows how  $UnTXD$  and  $UnRXD$  are internally connected in local loop-back mode. This mode is for testing the operation of a local UART module channel by sending data to the transmitter and checking data assembled by the receiver to ensure proper operations.



**Figure 27-22. Local Loop-Back**

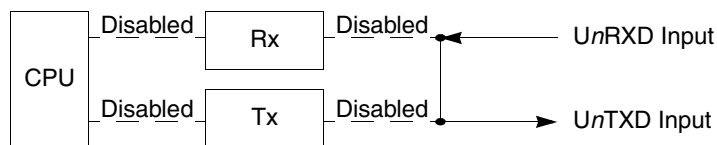
Features of this local loop-back mode are as follows:

- Transmitter and CPU-to-receiver communications continue normally in this mode.
- $UnRXD$  input data is ignored
- $UnTXD$  is held marking
- The receiver is clocked by the transmitter clock. The transmitter must be enabled, but the receiver need not be.

### 27.4.3.3 Remote Loop-Back Mode

In remote loop-back mode, shown in [Figure 27-23](#), the channel automatically transmits received data bit by bit on the  $UnTXD$  output. The local CPU-to-transmitter link is disabled. This mode is useful in testing receiver and transmitter operation of a remote channel. For this mode, the transmitter uses the receiver clock.

Because the receiver is not active, received data cannot be read by the CPU and all status conditions are inactive. Received parity is not checked and is not recalculated for transmission. Stop bits are sent as they are received. A received break is echoed as received until the next valid start bit is detected.



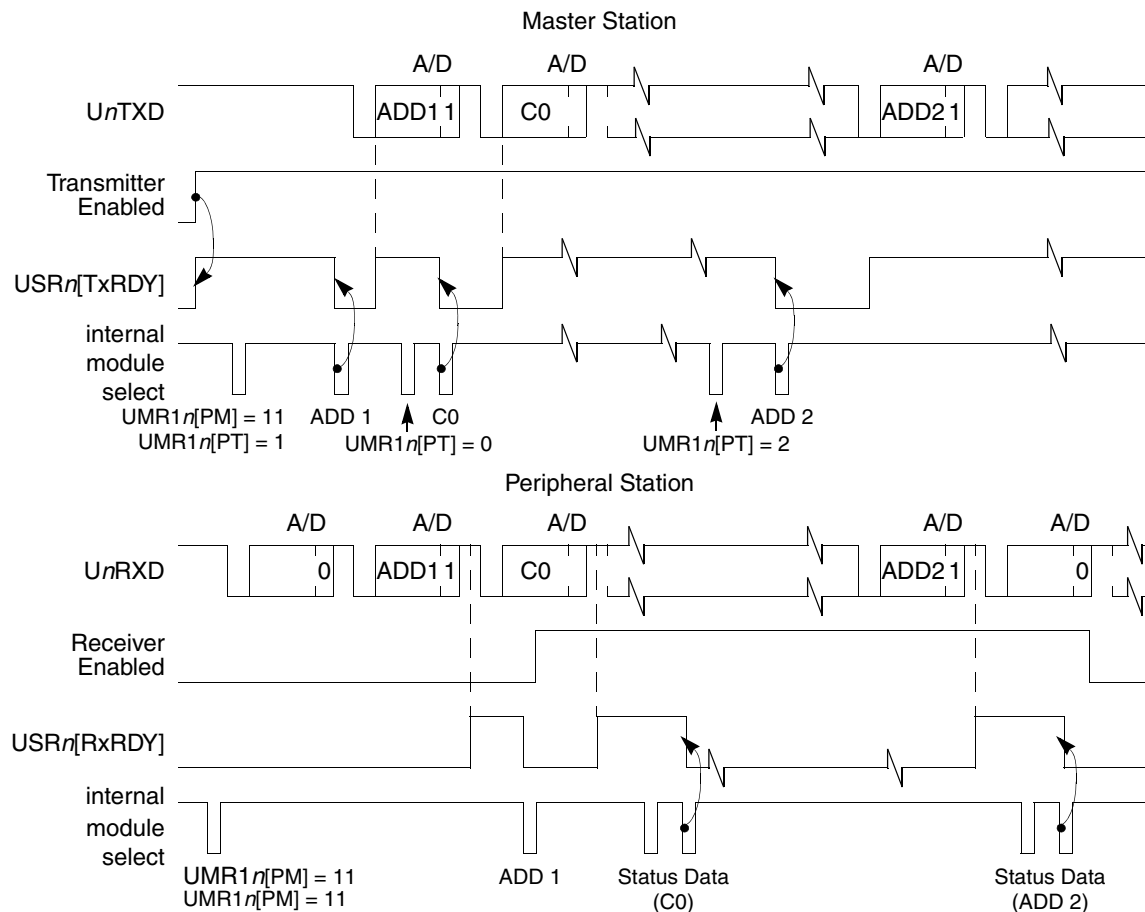
**Figure 27-23. Remote Loop-Back**

## 27.4.4 Multidrop Mode

Setting  $UMR1n[PM]$  programs the UART to operate in a wake-up mode for multidrop or multiprocessor applications. In this mode, a master can transmit an address character followed by a block of data characters targeted for one of up to 256 slave stations.

Although slave stations have their channel receivers disabled, they continuously monitor the master's data stream. When the master sends an address character, the slave receiver channel notifies its respective CPU by setting  $USRn[RxRDY]$  and generating an interrupt (if programmed to do so). Each slave station CPU then compares the received address to its station address and enables its receiver if it wishes to receive the subsequent data characters or block of data from the

master station. Slave stations not addressed continue monitoring the data stream. Data fields in the data stream are separated by an address character. After a slave receives a block of data, its CPU disables the receiver and repeats the process. Functional timing information for multidrop mode is shown in Figure 27-24.



**Figure 27-24. Multidrop Mode Timing Diagram**

A character sent from the master station consists of a start bit, a programmed number of data bits, an address/data (A/D) bit flag, and a programmed number of stop bits. A/D = 1 indicates an address character; A/D = 0 indicates a data character. The polarity of A/D is selected through UMR1n[PT]. UMR1n should be programmed before enabling the transmitter and loading the corresponding data bits into the transmit buffer.

In multidrop mode, the receiver continuously monitors the received data stream, regardless of whether it is enabled or disabled. If the receiver is disabled, it sets the RxRDY bit and loads the character into the receiver holding register FIFO provided the received A/D bit is a one (address tag). The character is discarded if the received A/D bit is zero (data tag). If the receiver is enabled, all received characters are transferred to the CPU through the receiver holding register during read operations.



In either case, the data bits are loaded into the data portion of the FIFO while the A/D bit is loaded into the status portion of the FIFO normally used for a parity error (USRn[PE]).

Framing error, overrun error, and break detection operate normally. The A/D bit takes the place of the parity bit; therefore, parity is neither calculated nor checked. Messages in this mode may still contain error detection and correction information. One way to provide error detection, if 8-bit characters are not required, is to use software to calculate parity and append it to the 5-, 6-, or 7-bit character.

## 27.4.5 Bus Operation

This section describes bus operation during read, write, and interrupt acknowledge cycles to the UART module.

### 27.4.5.1 Read Cycles

The UART module responds to reads with byte data. Reserved registers return zeros.

### 27.4.5.2 Write Cycles

The UART module accepts write data as bytes only. Write cycles to read-only or reserved registers complete normally without exception processing, but data is ignored.

## 27.4.6 Programming

The software flowchart, [Figure 27-25](#), consists of the following:

- UART module initialization—These routines consist of SINIT and CHCHK (See [Figure 27-25](#) and [Figure 27-26](#)). Before SINIT is called at system initialization, the calling routine allocates 2 words on the system FIFO. On return to the calling routine, SINIT passes UART status data on the FIFO. If SINIT finds no errors, the transmitter and receiver are enabled. SINIT calls CHCHK to perform the checks. When called, SINIT places the UART in local loop-back mode and checks for the following errors:
  - Transmitter never ready
  - Receiver never ready
  - Parity error
  - Incorrect character received
- I/O driver routine—This routine (See [Figure 27-28](#) and [Figure 27-29](#)) consists of INCH, the terminal input character routine which gets a character from the receiver, and OUTCH, which sends a character to the transmitter.
- Interrupt handling—Consists of SIRQ (See [Figure 27-28](#)), which is executed after the UART module generates an interrupt caused by a change-in-break (beginning of a break).

SIRQ then clears the interrupt source, waits for the next change-in-break interrupt (end of break), clears the interrupt source again, then returns from exception processing to the system monitor.

### 27.4.6.1 Interrupt and DMA Request Initialization

#### 27.4.6.1.1 Setting up the UART to Generate Core Interrupts

The list below gives the steps needed to properly initialize the UART to generate an interrupt request to the core.

1. Initialize ICRx register in the interrupt controller (ICR13 for UART0, ICR14 for UART1, and ICR15 for UART2).
2. Unmask appropriate bits in IMR in the interrupt controller (bits 13-15 for UART0-UART2 respectively).
3. Unmask appropriate bits in the core's Status Register (SR) to enable interrupts.
4. If TxRDY or RxRDY are being used to generate interrupt requests, then verify that DMAREQC (in the SCM) does not also assign the UART's TxRDY and RxRDY into DMA channels.
5. Initialize interrupts in the UART, see [Table 27-13](#).

**Table 27-13. UART Interrupts**

Register	Bit	Interrupt
UMR1x	6	RxIRQ
UIMRx	7	Change of State (COS)
UIMRx	2	Delta Break
UIMRx	1	RxFIFO Full
UIMRx	0	TxRDY

#### 27.4.6.1.2 Setting up the UART to Request DMA Service

The UART is capable of generating two different DMA request signals—transmit DMA requests and receive DMA requests.

The transmit DMA request signal is asserted when the TxRDY (transmitter ready) in the UART Interrupt Status Register, UISRn[TxRDY], is set. When the transmit DMA request signal is asserted, the DMA can initiate a data copy, reading the next character to be transmitted from memory and writing it into the UART transmit buffer (UTBn). This would allow the DMA channel to stream data from memory to the UART for transmission without processor intervention. Once the entire message has been moved into the UART, the DMA would typically generate an end-of-data-transfer interrupt request to the CPU. The resulting interrupt service routine (ISR) could query the UART programming model to determine the end-of-transmission status.

Similarly, the receive DMA request signal is asserted when the FFULL/RxRDY (FIFO full or receive ready) flag in the Interrupt Status Register, UISR[FFULL/RxRDY], is set. When the receive DMA request signal is asserted, the DMA can initiate a data move, reading the appropriate characters from the UART Receive Buffer (URBn) and storing them in memory. This allows the DMA channel to stream data from the UART receive buffer into memory without processor intervention. Once the entire message has been moved from the UART, the DMA would typically generate an end-of-data-transfer interrupt request to the CPU. The resulting interrupt service routine (ISR) should query the UART programming model to determine the end-of-transmission status. In typical applications, the receive DMA request should be configured to use RxRDY directly (and not FFULL) to remove any complications related to retrieving the final characters from the FIFO buffer.

The implementation described in this section allows independent DMA processing of transmit and receive data while still supporting interrupt notification to the processor for  $\overline{\text{CTS}}$  change-of-state and “delta break” error handling.

To configure the UART for DMA requests:

1. Initialize the DMAREQC in the SCM to map the desired UART DMA requests to the desired DMA channels. For example; setting DMAREQC[7:4] to 1000 maps UART0 receive DMA requests to DMA channel 1; setting DMAREQC[11:8] to 1101 maps UART1 transmit DMA requests to DMA channel 2; and so on. It is possible to independently map transmit based and receive based UART DMA requests in the DMAREQC.
2. Disable interrupts using the UIMR register. The appropriate UIMR bits must be cleared so that interrupt requests are disabled for those conditions for which a DMA request is desired. For example; to generate transmit DMA requests from UART1, then UIMR1[TxRDY] should be cleared. This will prevent TxRDY from generating an interrupt request while a transmit DMA request is generated.
3. Configure the GPACR and appropriate PACR registers located in the SCM for DMA access to IPSBAR space.
4. Initialize the DMA channel. The DMA should be configured for cycle steal mode and a source and destination size of one byte. This will cause a single byte to be transferred for each UART DMA request.

Table 27-14 shows the DMA requests.

**Table 27-14. UART DMA Requests**

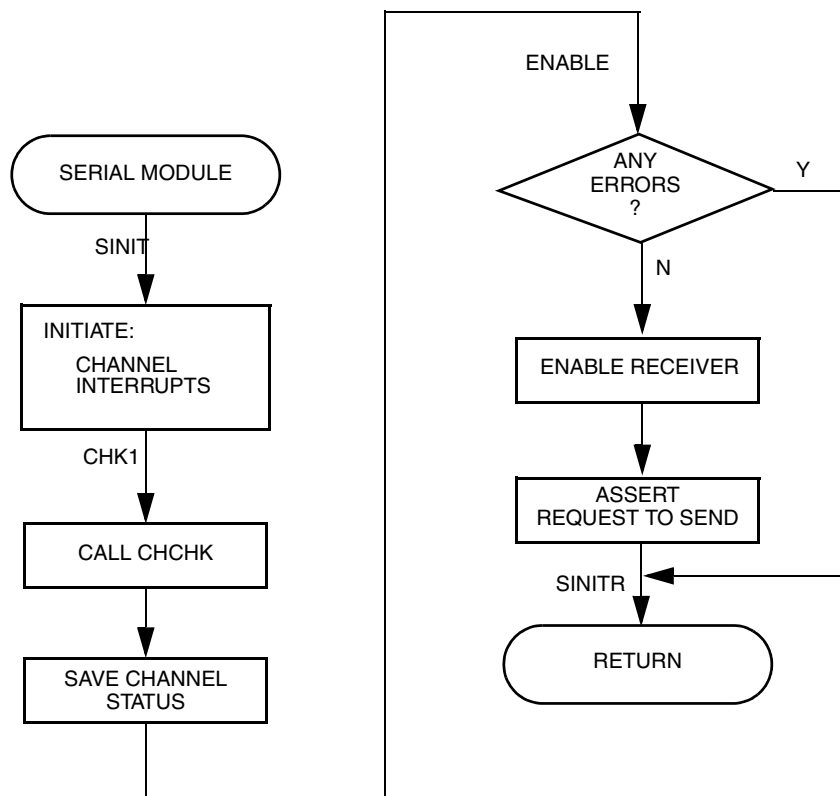
Register	Bit	DMA Request
UISRn	1	Receive DMA request
UISRn	0	Transmit DMA request

## 27.4.6.2 UART Module Initialization Sequence

Table 27-15 shows the UART module initialization sequence.

**Table 27-15. UART Module Initialization Sequence**

Register	Setting
UCR $n$	Reset the receiver and transmitter. Reset the mode pointer (MISC[2–0] = 0b001).
UIMR $n$	Enable the preferred interrupt sources.
UACR $n$	Initialize the input enable control (IEC bit).
UCSR $n$	Select the receiver and transmitter clock. Use timer as source if required.
UMR1 $n$	If preferred, program operation of receiver ready-to-send (RxRTS bit). Select receiver-ready or FIFO-full notification (RxRDY/FFULL bit). Select character or block error mode (ERR bit). Select parity mode and type (PM and PT bits). Select number of bits per character (B/Cx bits).
UMR2 $n$	Select the mode of operation (CMx bits). If preferred, program operation of transmitter ready-to-send (TxRTS). If preferred, program operation of clear-to-send (TxCTS bit). Select stop-bit length (SBx bits).
UCR $n$	Enable transmitter and/or receiver.



**Figure 27-25. UART Mode Programming Flowchart**

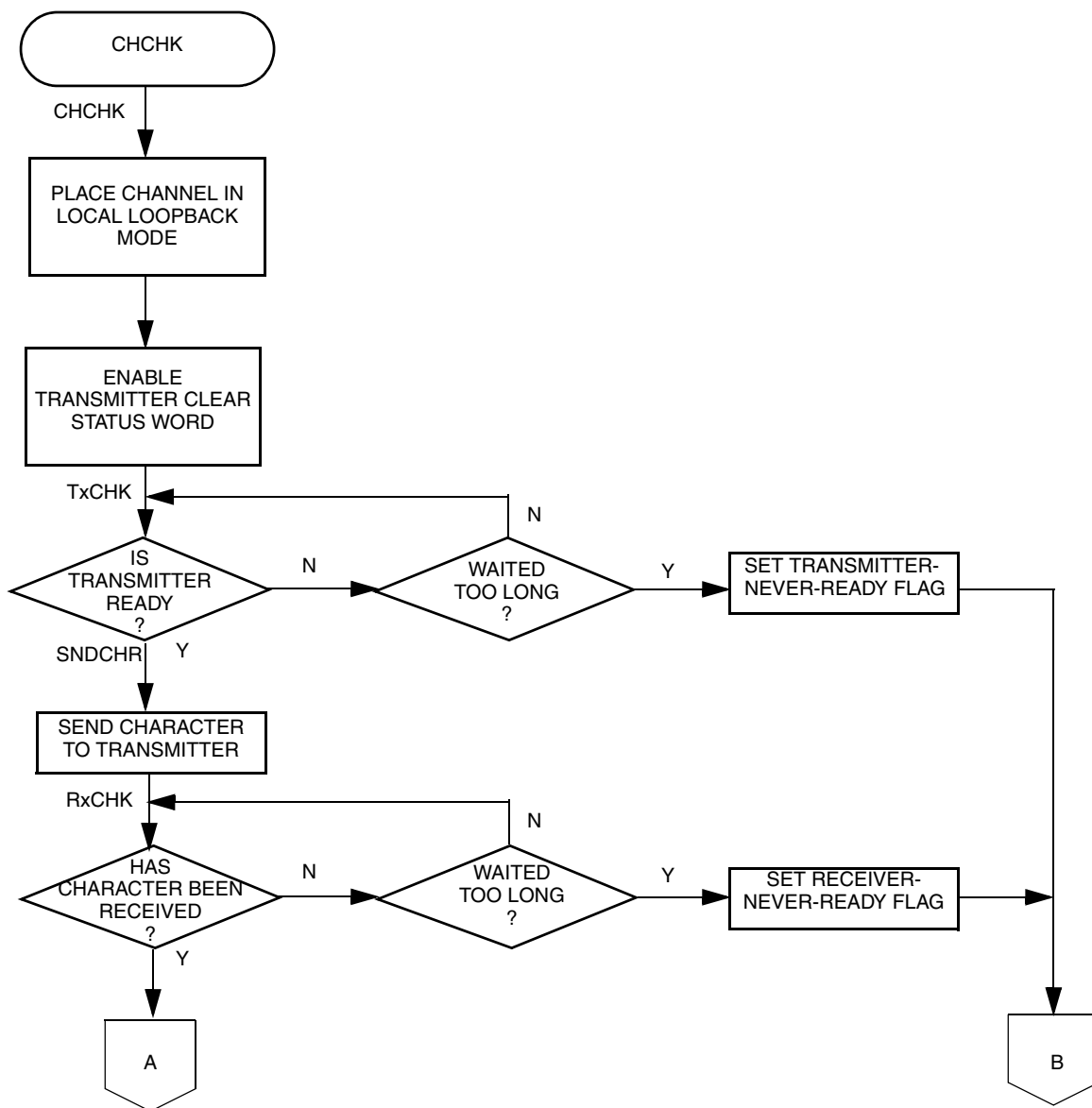
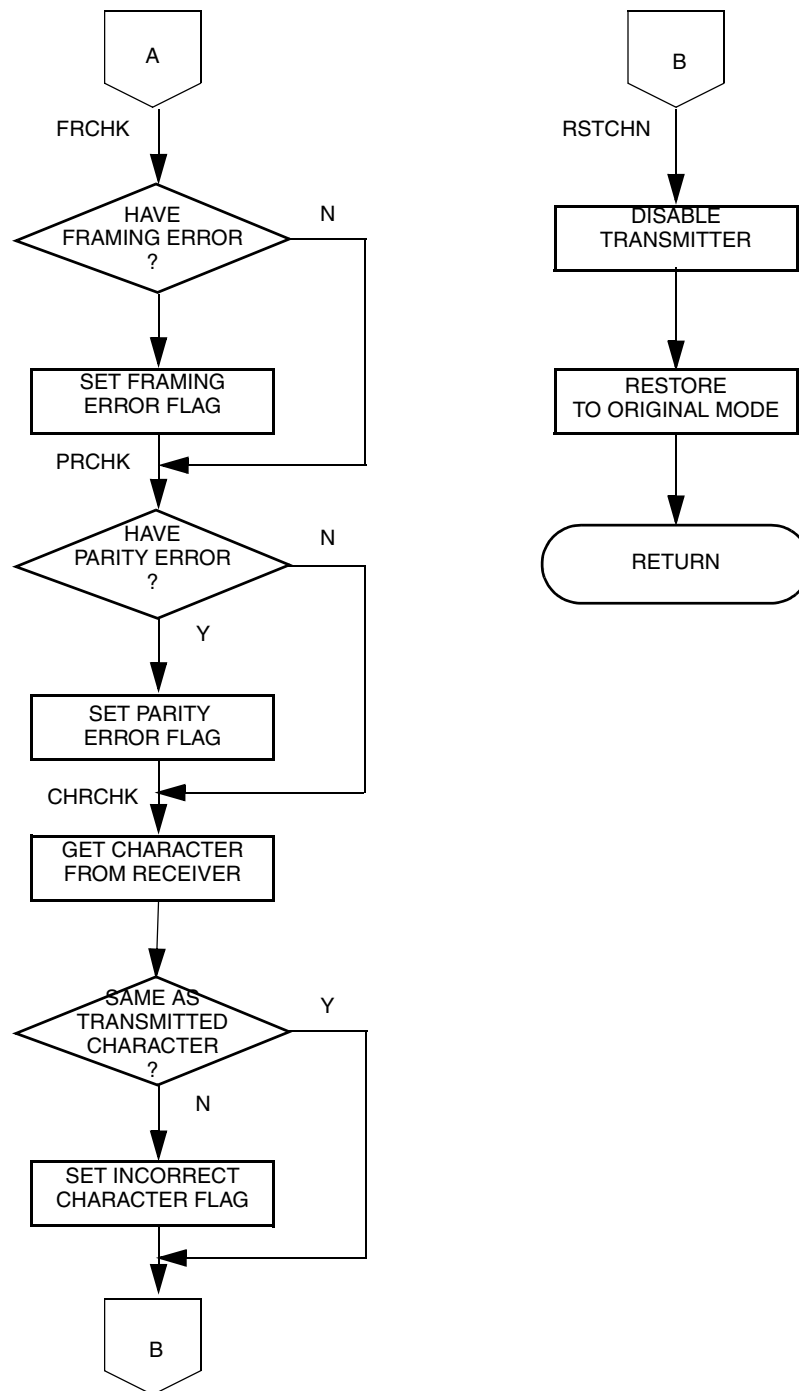
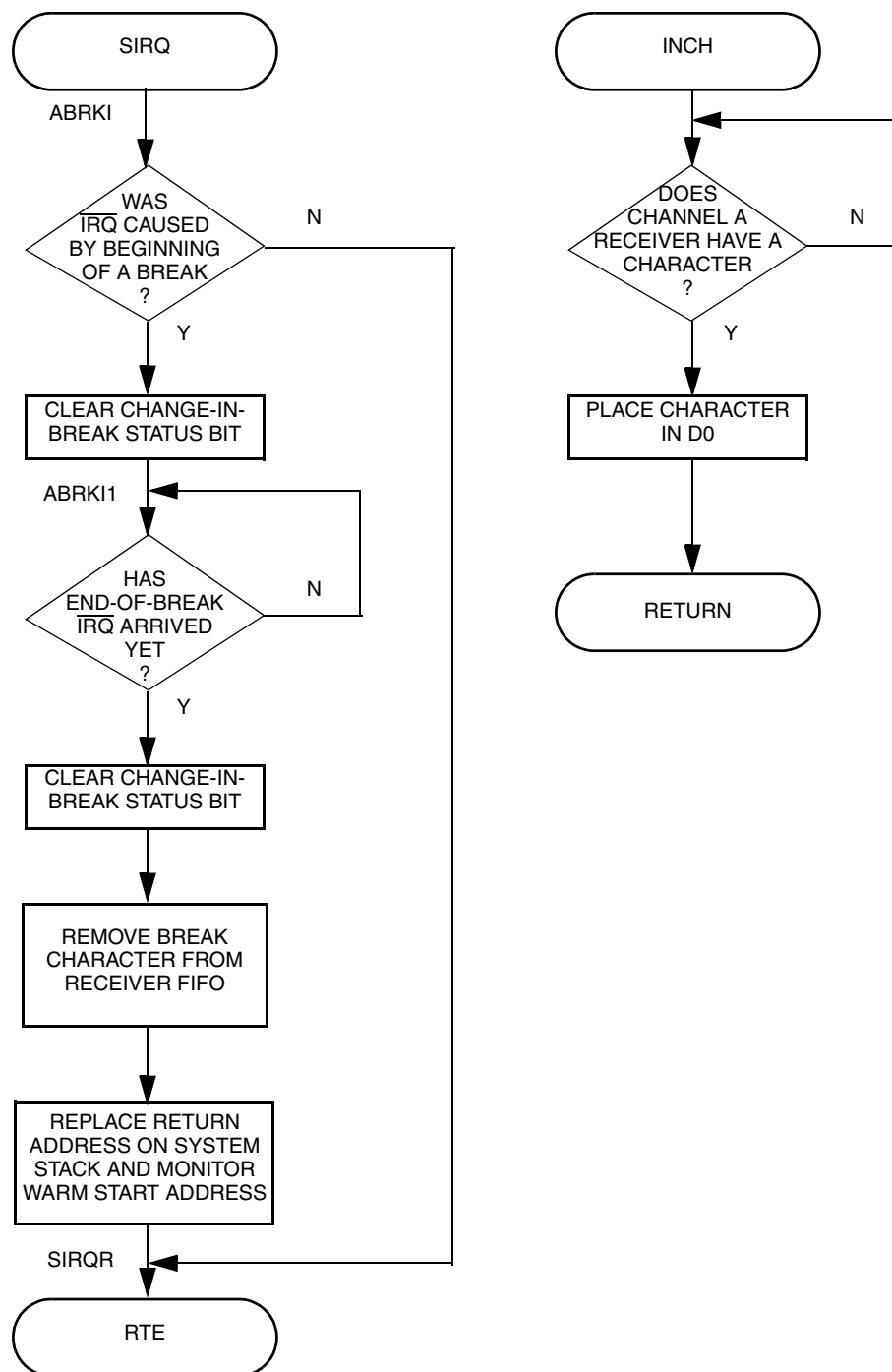


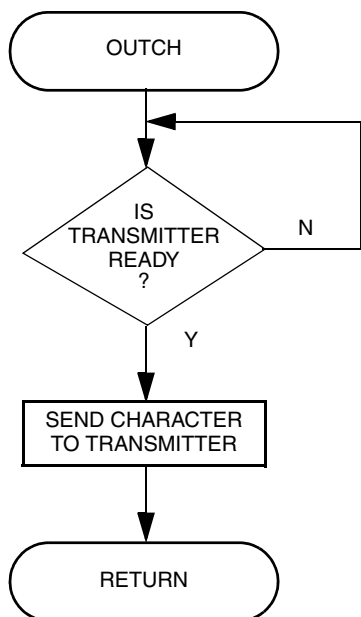
Figure 27-26. UART Mode Programming Flowchart (Continued)

**Figure 27-27. UART Mode Programming Flowchart (Continued)**



**Figure 27-28. UART Mode Programming Flowchart (Continued)**





**Figure 27-29. UART Mode Programming Flowchart (Continued)**



# Chapter 28

## Message Digest Hardware Accelerator (MDHA)

### 28.1 Introduction

This chapter describes the Message Digest Hardware Accelerator (MDHA).

#### 28.1.1 Overview

The MDHA is a hardware implementation of two of the world's most popular cryptographic hash functions: SHA-1 and MD5. SHA-1 and MD5 are both critical algorithms to the IPsec (IP Security) Protocol, a fast-spreading protocol for authentication and encryption over the internet in networking equipment. The MDHA also includes circuitry to automate the process of generating an HMAC (Hashed Message Authentication Code) as specified by RFC 2104 and EHMAC (Enhanced Hashed Message Authentication Code), using only the SHA-1 algorithm.

#### 28.1.2 Features

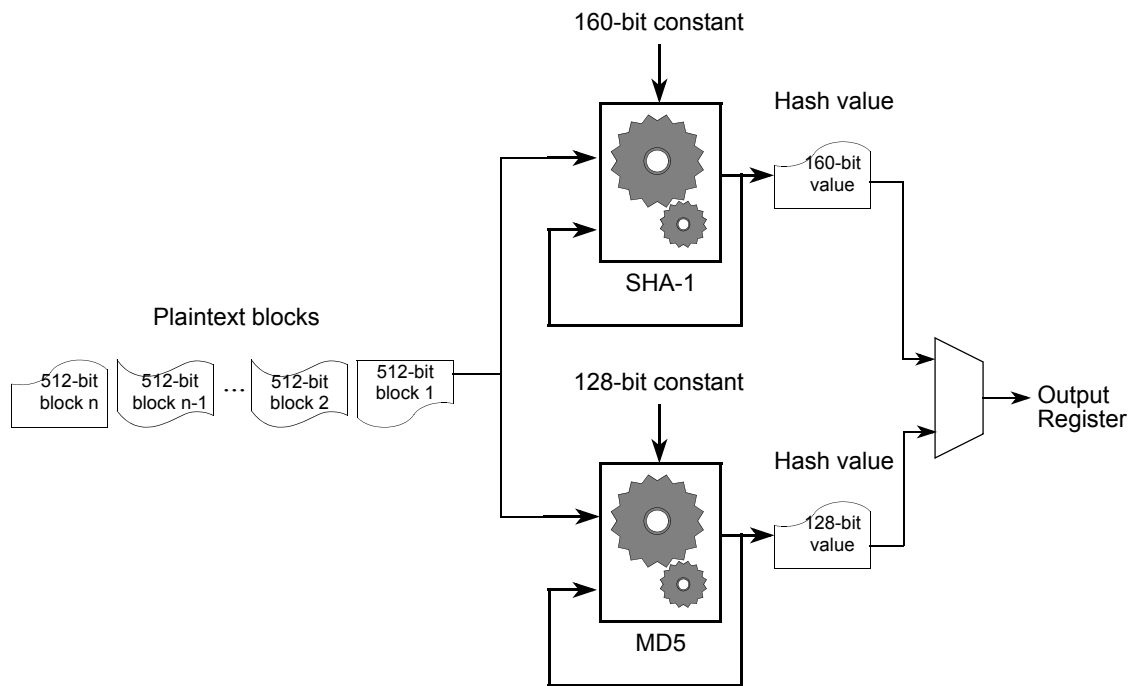
The MDHA computes a single message digest (or hash or integrity check) value of all the data presented on the input bus, using either the MD5 or SHA-1 algorithms for bulk data hashing. The MDHA includes these distinctive features:

- MD5 one way 128-bit hash function specified in RFC 1321.
- SHA-1 one way 160-bit hash function specified by the ANSI X9.30-2 and FIPS 180-1 standards.
- HMAC support for all algorithms, as specified in RFC 2104.
- EHMAC support for the SHA-1 algorithm.
- EHMAC key support up to 160 bits.
- Processes 512 bit blocks organized as  $16 \times 32$  bit longwords.
- Automatic Message and Key Padding.
- Internal 16x32 bit FIFO for temporary storage of hashing data.

With any hash algorithm, the larger message is mapped onto a smaller output space. Therefore collisions are potential, albeit not probable. The 160-bit hash value is a sufficiently large space such that collisions are extremely rare. The security of the hash function is based on the difficulty of locating collisions. That is, it is computationally infeasible to construct two distinct but similar messages that produce the same hash output.

This module is useful in many applications including hashing messages to generate digital signatures or computation of a shared secret. The digital signature is typically computed on a small input, however if the data to be signed is large, it is inefficient to sign the entire data. Instead, the large input data is hashed to a smaller value which is then signed. If the message is also sent to the verifying authority along with the signature, the verifying authority can verify the signature by recovering the hash value from the signature using the public key of the sender, hashing the message itself, and then comparing the computed hash value with the recovered hash value. If they match, then the verifying authority is confident that the data was signed by the owner of the private key that matches the public key, where the private key presumably is only known by the sender. This provides a measure of authentication and non-repudiation.

A conceptual block diagram of the MDHA module is shown in [Figure 28-1](#). Multiple input blocks are written to the MDHA module, and at the end, the hash value is read as the 160-bit output for SHA-1 or 128-bit output for MD5.



**Figure 28-1. MDHA Hashing Process**

### 28.1.3 Modes of Operation

- One-way hashing generation—a message digest, or a hash, is the result of a one-way function performed on a message. Two different hash functions can be used:
  - SHA-1: Secure Hash Algorithm defined in Federal Information Processing Standards Publication 180-1.

- MD5: Message Digest 5 Algorithm defined by Ron Rivest of MIT Laboratory for Computer Science and RSA Data Security, INC.
- MAC: A Message Authentication Code is a one-way function performed on a message with two user defined keys:
  - HMAC—Hashed message authentication can be used with either the SHA-1 or MD5 algorithm defined in Federal Information Processing Standards Publication 198.
  - EHMAC—Enhanced Hashed message authentication can only be used with the SHA-1 algorithm.

## 28.2 Memory Map/Register Definition

Table 28-1 shows the MDHA memory map.

**Table 28-1. MDHA Module Memory Map**

IPSBAR Offset	Mnemonic	[31:24]	[23:16]	[15:8]	[7:0]	Access
0x19_0000	MDMR	MDHA Mode Register				R/W
0x19_0004	MDCR	MDHA Control Register				R/W
0x19_0008	MDCMR	MDHA Command Register				W
0x19_000C	MDSR	MDHA Status Register				R
0x19_0010	MDISR	MDHA Interrupt Status Register				R
0x19_0014	MDIMR	MDHA Interrupt Mask Register				R/W
0x19_001C	MDDSR	MDHA Data Size Register				R/W
0x19_0020	MDIN	MDHA Input FIFO Register				R/W
0x19_0030	MDA0	Message Digest A0 Register				R/W
0x19_0034	MDB0	Message Digest B0 Register				R/W
0x19_0038	MDC0	Message Digest C0 Register				R/W
0x19_003C	MDD0	Message Digest D0 Register				R/W
0x19_0040	MDE0	Message Digest E0 Register				R/W
0x19_0044	MDMDS	Message Data Size Register				R/W
0x19_0048– 0x19_006F	—	Reserved				—
0x19_0070	MDA1	Message Digest A1 Register				R/W
0x19_0074	MDB1	Message Digest B1 Register				R/W
0x19_0078	MDC1	Message Digest C1 Register				R/W
0x19_007C	MDD1	Message Digest D1 Register				R/W
0x19_0080	MDE1	Message Digest E1 Register				R/W

### 28.2.1 MDHA Mode Register (MDMR)

The MDMR stores the current processing mode. It can be written before a hashing operation begins. Once the hashing operation has begun an error will be generated if the register is written. This register is reset only via a hardware or software reset.

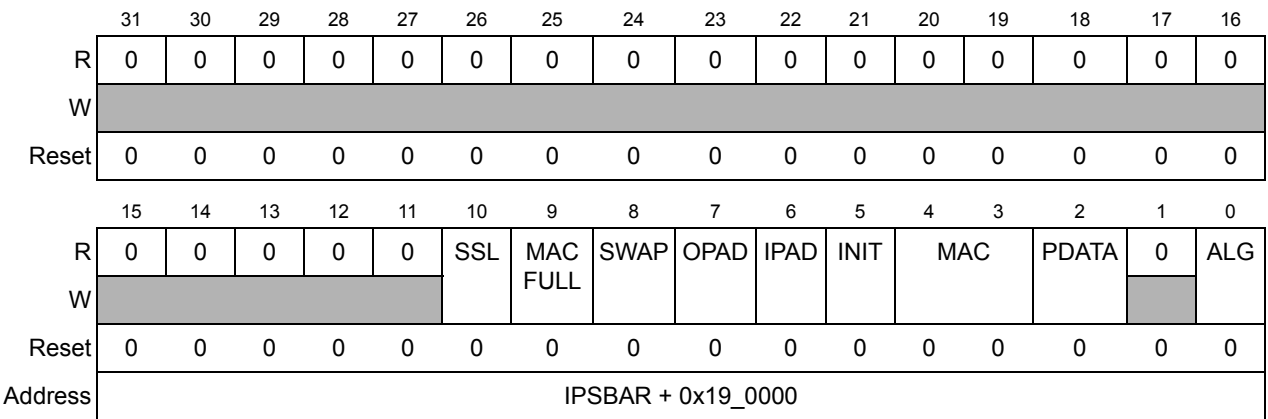


Figure 28-2. MDHA Mode Register (MDMR)

Table 28-2. MDMR Field Descriptions

Bits	Name	Description
31–11	—	Reserved, should be cleared.
10	SSL	Secure socket layer MAC. Implements the SSL defined MAC. Only applicable for the MD5 algorithm. 0 Do not perform SSL. 1 Perform SSL
9	MACFULL	Message authentication code full. Allows the user to input a key and message data and have the accelerator do the complete MAC in one step. Used directly with HMAC or EHMAC mode. 0 Do not perform MAC FULL. 1 Perform MAC FULL.

**Table 28-2. MDMR Field Descriptions (Continued)**

Bits	Name	Description																																																								
8	SWAP	<p>Swap message digest. For SHA-1 only. Swap the output direction of the Message Digest data. The data registers are reversed and byte swapped. This allows for viewing data in the reverse order which might be used by other algorithms. See the table below for an example.</p> <p>0 Do not perform swap. 1 Swap output direction.</p> <table><tr><th>Algorithm</th><th>Message Digest Register</th><th colspan="4">SWAP = 0</th><th colspan="4">SWAP = 1</th></tr><tr><td rowspan="5">SHA-1</td><td>A0</td><td>D</td><td>C</td><td>B</td><td>A</td><td>Q</td><td>R</td><td>S</td><td>T</td></tr><tr><td>B0</td><td>H</td><td>G</td><td>F</td><td>E</td><td>M</td><td>N</td><td>O</td><td>P</td></tr><tr><td>C0</td><td>L</td><td>K</td><td>J</td><td>I</td><td>I</td><td>J</td><td>K</td><td>L</td></tr><tr><td>D0</td><td>P</td><td>O</td><td>N</td><td>M</td><td>E</td><td>F</td><td>G</td><td>H</td></tr><tr><td>E0</td><td>T</td><td>S</td><td>R</td><td>Q</td><td>A</td><td>B</td><td>C</td><td>D</td></tr></table>	Algorithm	Message Digest Register	SWAP = 0				SWAP = 1				SHA-1	A0	D	C	B	A	Q	R	S	T	B0	H	G	F	E	M	N	O	P	C0	L	K	J	I	I	J	K	L	D0	P	O	N	M	E	F	G	H	E0	T	S	R	Q	A	B	C	D
Algorithm	Message Digest Register	SWAP = 0				SWAP = 1																																																				
SHA-1	A0	D	C	B	A	Q	R	S	T																																																	
	B0	H	G	F	E	M	N	O	P																																																	
	C0	L	K	J	I	I	J	K	L																																																	
	D0	P	O	N	M	E	F	G	H																																																	
	E0	T	S	R	Q	A	B	C	D																																																	
7	OPAD	<p>Outer padding of message. Exclusive OR the message with 0x5C5C_5C5C. Hash used with HMAC. Requires key to be loaded into the FIFO</p> <p>0 Do not perform padding 1 Perform padding</p>																																																								
6	IPAD	<p>Inner padding of message. Exclusive OR the message with 0x3636_3636. Hash used with HMAC. Requires key to be loaded into the FIFO</p> <p>0 Do not perform padding 1 Perform padding</p>																																																								
5	INIT	<p>Initialization. Performs algorithm specific initialization of the digest registers. Most operation will require this bit to be set. Only static operations that are continuing from a known intermediate hash value should clear this bit.</p> <p>0 Do not perform initialization 1 Initialize the selected algorithm's starting registers</p>																																																								
4-3	MAC	<p>Message authentication code. Performs message authentication on messages. Requires keys loaded into the context and key registers.</p> <p>00 Do not perform MAC 01 Perform HMAC 10 Perform EHMAC 11 Reserved</p>																																																								
2	PDATA	<p>Pad data bit. Performs automatic message padding on the current partial message block.</p> <p>0 Do not perform padding 1 Perform padding</p>																																																								
1	—	Reserved, should be cleared.																																																								
0	ALG	<p>Algorithm. Selects which algorithm the MDHA module uses</p> <p>0 Secure Hash Algorithm (SHA-1) 1 Message Digest 5 (MD5)</p>																																																								

### 28.2.1.1 Invalid Modes

The following mode combinations will trigger an interrupt to the interrupt controller and set the mode error bit in the MDHA interrupt status register.

**Table 28-3. Invalid MDMR Bit Settings**

MDMR bit settings		Comments
Setting any reserved bits.		—
IPAD=1	OPAD=1	Asserting both of the signals at the same time causes a mode to occur that the MDHA is not capable of performing. These two modes must be performed separately.
IPAD=1	PDATA=1	According to HMAC and EHMAC standards no padding is done to the data when the IPAD function is performed.
OPAD=1	PDATA=1	According to HMAC and EHMAC standards no padding is done to the data when the OPAD function is performed.
MAC=10 (EHMAC)	IPAD=1	MDHA requires that the IPAD step be performed as a separate hash operation than message authentication.
MAC=10 (EHMAC)	OPAD=1	MDHA requires that the OPAD step be performed as a separate hash operation than message authentication.
MAC=10 (EHMAC)	ALG=1 (MD5)	The standard for EHMAC is only defined for the SHA-1 algorithm.
MAC=01 (HMAC)	IPAD	MDHA requires that the IPAD step be performed as a separate hash operation than message authentication.
MAC=01 (HMAC)	OPAD	MDHA requires that the OPAD step be performed as a separate hash operation than message authentication.
SSL=1	ALG=0 (SHA-1)	SSL MAC is only functional for the MD5 algorithm.
SWAP=1	ALG=1 (MD5)	The SWAP bit is designed to support a particular function for the SHA-1 algorithm and is invalid for MD5.

### 28.2.2 MDHA Control Register (MDCR)

The MDCR contains bits that should be set following a hardware reset.



	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
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
	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	IE
W																
Reset	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
Address	IPSBAR + 0x19_0004															

Figure 28-3. MDHA Control Register (MDCR)

Table 28-4. MDCR Field Descriptions

Bits	Name	Description
31–1	—	Reserved
0	IE	Interrupt enable. Enables/Disables interrupts from the MDHA module. 0 Disable interrupt 1 Enable interrupt

### 28.2.3 MDHA Command Register (MDCMR)

The MDCR will always read zeros. The bits in the command register are used for software control of hashing and resetting.

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
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
	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													GO	CI	RI	SWR
Reset	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
Address	IPSBAR + 0x19_0008															

Figure 28-4. MDHA Command Register (MDCMR)

**Table 28-5. MDCMR Field Descriptions**

Bits	Name	Description
31–4	—	Reserved
3	GO	Go. Indicates that all data has been loaded into the input FIFO and the module should complete all processing. This bit is self clearing. 0 Do not complete all processing. 1 Finish all processing.
2	CI	Clear IRQ. Clears errors in the MDISR register and deasserts any interrupt requests from the MDHA module. This bit is self clearing. 0 Do not clear interrupts & errors. 1 Clear interrupts & errors.
1	RI	Re-initialize. Re-initializes memory and clears all registers except the MDESM and MDCR. 0 No re-initialization 1 Re-initialize the MDHA module
0	SWR	Software reset. Resets all registers and re-initialize memory of the MDHA. Functionally equivalent to hardware reset. This bit is self clearing. 0 No reset 1 Software reset

## 28.2.4 MDHA Status Register (MDSR)

The MDSR stores the current status of the MDHA. This register is used to debug errors and to give a view into the workings of the MDHA's internal engines.

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	IFL							
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
Field	APD			0	0	FS			GNW	HSH	0	BUSY	RD	ERR	DONE	INT
W																
Reset	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0
Address	IPSBAR + 0x19_000C															

**Figure 28-5. MDHA Status Register (MDSR)****Table 28-6. MDSR Field Descriptions**

Bits	Name	Bit Description
31–24	—	Reserved, should be cleared.
23–16	IFL	Input FIFO level. Read-only. The current number of longwords that are in the Input FIFO. IFL will range from 0–16 longwords (0x00-0x10).

**Table 28-6. MDSR Field Descriptions (Continued)**

Bits	Name	Bit Description
15–13	APD	Auto pad state. Read-only. Indicates the current state of the autopadder for debug purposes. 000 Perform standard auto padding. 001 Pad last word. 010 Add a word for padding. 011 Last hash for the EHMAC. 100 Stall state (Auto Padder will pass no data to engine). This is the default state that the module enters whenever there is an error. All other settings are reserved.
12–11	—	Reserved, should be cleared.
10–8	FS	FIFO size. Read-only. Indicates the size of the internal FIFO, which is fixed at 16 longwords for the MCF5275. 100 16 longwords
7	GNW	Get next word. Read-only. Indicates that the MDHA engine has not filled an entire block and is requesting more data. 0 Does not need any data 1 Requesting more data
6	HSB	Hashing. Read-only. Indicates that data is currently being hashed 0 Waiting for more data 1 Hashing current data
5	—	Reserved, should be cleared.
4	BUSY	Busy. Read-only. Indicates that the module is busy processing data. 0 Idle or done 1 Busy processing data
3	RD	Reset interrupt. Read-only. Indicates the MDHA module has completed resetting. 0 Reset in progress 1 Completed reset sequence
2	ERR	Error interrupt. Read-only. Indicates that an error has occurred. Set if any bit in the MDISR is set. 0 No Error 1 Error has occurred
1	DONE	Done interrupt. Read-only. Indicates that the MDHA module has completed processing the requested amount of data. 0 Not complete 1 Done processing
0	INT	MDHA single interrupt. Read-only. Indicates that either the MDHA module has finished processing the message and the hash result is ready to be read from the message digest register or there is an error. 0 No interrupt 1 Done or error interrupt

## 28.2.5 MDHA Interrupt Status & Mask Registers (MDISR and MDIMR)

The MDISR describes the current error that has taken place. An interrupt request is generated when any one of these bits is set unless the corresponding bit is set in the MDIMR. MDISR is only cleared after a hardware or software reset has been performed. The bit definitions for MDISR and MDIMR are similar.

### NOTE

Masking errors should only be used for debug purposes. A masked error will most likely cause invalid data.

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
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
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	GTDS	ERE	RMDP	0	DSE	IME	0	NEIF	0	IFO
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Address	IPSBAR + 0x19_0010															

**Figure 28-6. MDHA Interrupt Status & Mask Registers (MDISR and MDIMR)**

**Table 28-7. MDISR & MDIMR Field Descriptions**

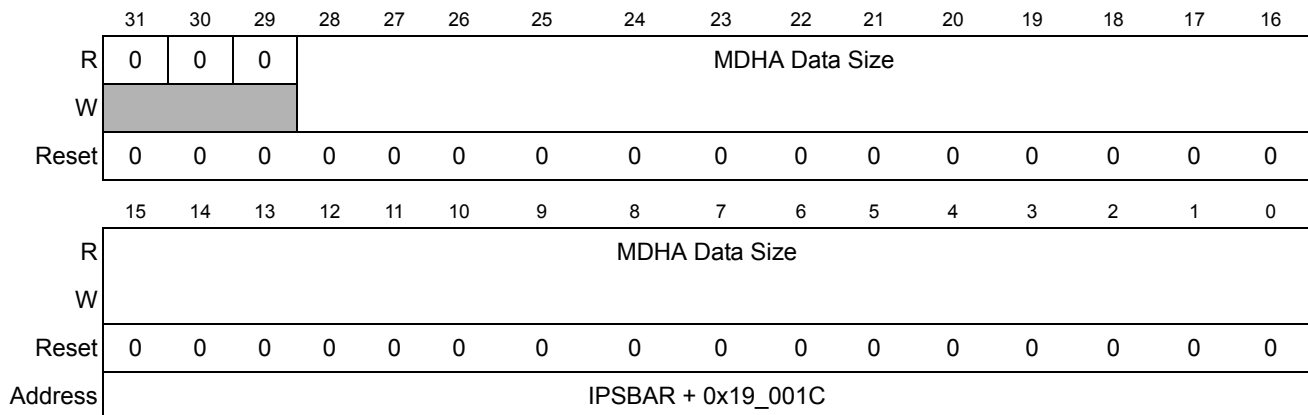
Bits	Name	Bit Description
31–10	—	Reserved, should be cleared.
9	GTDS	Greater than data size error. Read only. Indicates that the GO bit was set in the MDCR and the data size written to the MDSSR is greater than the amount of data written to the FIFO. 0 No error 1 Datasize is greater than the message size
8	ERE	Early read error. Read only. A context register was read from while the module was busy processing data. 0 No error 1 Early read error
7	RMDP	Register modified during processing. Read only. An MDHA register was modified while the module was busy processing data. 0 No error 1 Register modified
6	—	Reserved, should be cleared.

**Table 28-7. MDISR & MDIMR Field Descriptions (Continued)**

Bits	Name	Bit Description
5	DSE	Illegal data size. Read only. Illegal data size was written to the MDHA Data Size Register (MDDSR). Data size written into the MDDSR is greater than the allocated size. 0 No error 1 Illegal data size in MDDSR
4	IME	Illegal mode interrupt. Read only. Illegal mode is set in the MDMR. Consult <a href="#">Section 28.2.1.1, "Invalid Modes,"</a> for more information on invalid modes. 0 No error 1 Illegal value in MDMR
3	—	Reserved, should be cleared.
2	NEIF	Non-empty input FIFO upon done. Read only. The Input FIFO contained data when processing was completed 0 No error 1 FIFO contained data when finished processing
1	—	Reserved
0	IFO	Input FIFO Overflow. Read only. The Input FIFO has been written to while full. 0 No overflow occurred 1 Input FIFO overflow error

### 28.2.6 MDHA Data Size Register (MDDSR)

The MDDSR stores the size of the last block of data to be processed. This value is in bytes. The first two bits are used to identify the ending byte location in the last word. This is used to add the data padding when auto padding is selected in the MDMR. Load this register with the amount of data to be processed in the FIFO. This register is cleared when the MDHA is reset, re-initialized and at the end of processing the complete message.

**Figure 28-7. MDHA Data Size Register (MDDSR)**

## 28.2.7 MDHA Input FIFO (MDIN)

The MDIN provides temporary storage for data to be used during hashing. The FIFO is a write only register and attempting to read from this register will always return 0. If the FIFO is written to when the FIFO Level is full then an interrupt request is generated and the MDISR[IFO] bit will be set. The MDSR[IFL], described in [Section 28.2.4, “MDHA Status Register \(MDSR\),”](#) can be polled to monitor how many 32-bit longwords are currently resident in the FIFO.

## 28.2.8 MDHA Message Digest Registers 0 (MDx0)

The MDHA message digest registers 0 consist of five 32-bit registers (MDA0, MDB0, MDC0, MDD0, and MDE0). These registers store the five (SHA-1) or four (MD5) 32-bit longwords that are the final answer (digest/context) of the hashing process. Message digest data may only be read if the MDSR[DONE] bit is set. Any reads prior to this result is an early read error (MDISR[ERE]). The message digest registers will always return all zeros when an error is generated. Each word (4 bytes) in the MDx0 is assumed to be in little endian byte order for all reads/writes. All corrections will be done internal. This register is cleared when the MDHA is reset or re-initialized. The reset values for the registers are the algorithms defined chaining variable values.

	31	0
R	MDA0, MDB0, MDC0, MDD0, MDE0	
W		
Reset	0x0123_4567 (MDA0); 0x89AB_CDEF (MDB0); 0xFEDC_BA98 (MDC0); 0x7654_3210 (MDD0); 0xF0E1_D2C3 (MDE0)	
Address	IPSBAR + 0x19_0030 (MDA0); IPSBAR + 0x19_0034 (MDB0); IPSBAR + 0x19_0038 (MDC0); IPSBAR + 0x19_003C (MDD0); IPSBAR + 0x19_0040 (MDE0)	

**Figure 28-8. MDHA Message Digest Registers 0 (MDx0)**

## 28.2.9 MDHA Message Data Size Register (MDMDS)

The MDMDS is a 32-bit register which, when read, will store the size of the current hash operation. This register is also used to write in the data size from a resumed hash operation. This data size will be added to the MDDSR to complete the auto pad step.

	31	0
R	Message Data Size	
W		
Reset	0 0	
Address	IPSBAR + 0x19_0044	

**Figure 28-9. MDHA Message Data Size Register (MDMDS)**



### 28.3.1 MDHA Top Control

The MDHA Top Control block enables the FIFO whenever the MDHA module is enabled. This block also captures both error and done status from the MDHA Logic block and generates a single interrupt to the interrupt controller.

### 28.3.2 FIFO

The FIFO block contains a  $16 \times 32$ -bit FIFO that is used for temporary storage of the data to be hashed.

### 28.3.3 MDHA Logic

The MDHA logic block consists of 7 sub-blocks: the address decoder, interface control, auto-padder, auto-padder control, algorithm engine, algorithm engine control, and status interrupt as shown in [Figure 28-11](#).

#### 28.3.3.1 Address Decoder

The address decoder drives out the proper data from the module or captures incoming data into the appropriate register.

#### 28.3.3.2 Interface Control

The interface block decodes the MDMR and outputs all control signals to all other blocks. Control signals are received from other modules to send a pop signal to the FIFO.

#### 28.3.3.3 Auto-Padder

The Auto-padder takes longwords in from the FIFO and then either passes it directly to the engine or pads the word according to the control bits that are set. The IPAD and OPAD is done to all longwords in this block before they are passed directly to the engine. This block takes care of passing the proper data to the engine for the EHMAC mode of operation. This is done by an internal counter that will leave 351 bits in the Input FIFO.

#### 28.3.3.4 Hashing Engine

This module is the core of the Message Digest Hardware Accelerator that is capable of computing the Secure Hash Algorithm (SHA-1) or Message Digest 5 (MD5).



### 28.3.3.5 Hashing Engine Control

This module is the control unit of the MDHA that is capable of computing the Secure Hash Algorithm (SHA-1) and Message Digest 5 algorithm (MD5). This module keeps track of all rounds and tells the rest of the module when the operation has been completed.

### 28.3.3.6 Status Interrupt

This block generates the error interrupt if the host performs an illegal operation. The cause of the error is flagged in the MDISR ([Section 28.2.5, “MDHA Interrupt Status & Mask Registers \(MDISR and MDIMR\)”](#)). If an error occurs, the MDHA core engine is halted. This prevents the core from continuing operation with invalid data.

## 28.4 Initialization/Application Information

### 28.4.1 Performing a Standard HASH Operation

1. Reset the MDHA using the MDCMR[SWR] bit.
2. MDCR register write. Enable the interrupts. (optional)
3. MDMR register write. Select algorithm, data padding, and algorithm initialization.
4. Write message data into the FIFO in longwords.
5. MDDSR register write. Load this register with the length of the message data (without padding) in bytes.
6. Set MDCMR[GO].
7. Wait for MDSR[INT] to be set or done interrupt to be triggered to indicate successful completion (or failure).

#### NOTE

You will need to provide a time-out feature in your interrupt handler. The MDHA will stall with no response if it is waiting for message data. This will most likely occur if the MDDSR write is not received or auto-padding is disabled and a partial message block is provided.

8. If MDSR[DONE] is set or done interrupt is triggered, then read the message digest from the message digest registers.

### 28.4.2 Performing a HMAC Operation Without the MACFULL Bit

The HMAC is done in three separate steps without the MACFULL bit. Each step requires the reinitialization of the MDHA.

### 28.4.2.1 Generation of Key with IPAD

1. Reset the MDHA using the MDCMR[SWR] bit.
2. MDCR register write. Enable the interrupts. (optional)
3. MDMR register write. Select algorithm, IPAD, and MD initialization.
4. The data FIFO is filled with the key.
5. MDDSR register write. Load this register with the length of the key (without padding) in bytes.
6. MDHA does the required IPAD of key.
7. MDHA does the required algorithm's auto padding of message.
8. Set the MDCMR[GO] bit.
9. Wait for MDSR[INT] to be set or done interrupt to be triggered to indicate successful completion (or failure).

#### NOTE

You will need to provide a time-out feature in your interrupt handler. The MDHA will stall with no response if it is waiting for message data. This will most likely occur if the MDDSR write is not received or auto-padding is disabled and a partial message block is provided.

10. If MDSR[DONE] is set or done interrupt is triggered, then read the message digest and the message digest count from the message digest registers.

### 28.4.2.2 Generation of Key with OPAD

1. Reset the MDHA using the MDCMR[SWR] bit.
2. MDCR register write. Enable the interrupts. (optional)
3. MDMR register write. Select algorithm, OPAD, and MD initialization.
4. The data FIFO is filled with the key.
5. MDDSR register write. Load this register with the length of the key (without padding) in bytes.
6. MDHA does the required OPAD of key.
7. MDHA does the required algorithm's auto padding of message.
8. Set the MDCMR[GO] bit.
9. Wait for MDSR[INT] to be set or done interrupt to be triggered to indicate successful completion (or failure).

**NOTE**

You will need to provide a time-out feature in your interrupt handler. The MDHA will stall with no response if it is waiting for message data. This will most likely occur if the MDDSR write is not received or auto-padding is disabled and a partial message block is provided.

10. If MDSR[**DONE**] is set or done interrupt is triggered, then read the message digest and the message digest count from the message digest registers.

**28.4.2.3 HMAC Hash**

1. Reset the MDHA using the MDCMR[**SWR**] bit.
2. MDCR register write. Enable the interrupts. (optional)
3. MDMR register write. Select algorithm, data padding, and HMAC.
4. MDMD5 register write. Load this register with the length of the message from the OPAD step.
5. Direct context load of (OPAD XOR key) into MDx1 registers from the OPAD step in the previous section.
6. Direct context load of (IPAD XOR key) into MDx0 registers from the IPAD step.
7. Direct context load of MDMD5 register from the IPAD step (this should be 64 bytes).
8. Fill data FIFO with message to be hashed.
9. MDDSR register write. Load this register with the length of the message data (without padding) in bytes.
10. MDHA does the required algorithm's auto padding of the message.
11. Set the MDCMR[**GO**] bit.
12. Wait for MDSR[**INT**] to be set or done interrupt to be triggered to indicate successful completion (or failure).

**NOTE**

You will need to provide a time-out feature in your interrupt handler. The MDHA will stall with no response if it is waiting for message data. This will most likely occur if the MDDSR write is not received or auto-padding is disabled and a partial message block is provided.

13. If MDSR[**DONE**] is set or done interrupt is triggered, then read the message digest from the message digest registers.

**28.4.3 Performing a SHA-1 EHMAC**

EHMAC can only be performed with the SHA-1 algorithm. The Enhanced HMAC requires the keys to be hashed with IPAD and OPAD prior to the step below. The IPAD and OPAD step can be

followed in [Section 28.4.2.1, “Generation of Key with IPAD”](#) and [Section 28.4.2.2, “Generation of Key with OPAD.”](#)

1. Reset the MDHA using the MDCMR[SWR] bit.
2. MDCR register write. Enable the interrupts.
3. MDMR register write. Select algorithm, data padding, and EHMAC.
4. Direct context load of (OPAD XOR key) into MDx1 registers from OPAD step.
5. MDMD5 register write. Load this register with the length of the message from the OPAD step.
6. Direct context load of (IPAD XOR key) into MDx0 registers from IPAD step.
7. Direct context load of MDMD5 register from the IPAD step (this should be 64 bytes).
8. Fill data FIFO with message to be hashed.
9. MDDSR register write. Load this register with the length of the message data (without padding) in bits.
10. MDHA does the required algorithm’s auto padding of the message.
11. Set the MDCMR[GO] bit.
12. Wait for MDSR[DONE] to be set or done interrupt to be triggered to indicate successful completion (or failure).

#### NOTE

You will need to provide a time-out feature in your interrupt handler. The MDHA will stall with no response if it is waiting for message data. This will most likely occur if the MDDSR write is not received or auto-padding is disabled and a partial message block is provided.

13. If MDSR[DONE] is set or done interrupt is triggered, then read the message digest from the message digest registers.

## 28.4.4 Performing a MAC Operation With the MACFULL Bit

The HMAC/EHMAC is done in one step with the MACFULL bit.

1. Reset the MDHA using the MDCMR[SWR] bit.
2. MDCR register write. Enable the interrupts. (optional)
3. MDMR register write. Select algorithm, data padding, HMAC or EHMAC, and MACFULL bits.
4. Direct context load of key into MDx1 registers.
5. MDMD5 register write. Load this register with the length of the key.
6. Fill data FIFO with message to be hashed.

7. MDDSR register write. Load this register with the length of the message data (without padding) in bytes.
8. MDHA does the required algorithm's auto padding of the message.
9. Set the MDCMR[GO] bit.
10. Wait for MDSR[INT] to be set or done interrupt to be triggered to indicate successful completion (or failure).

### NOTE

You will need to provide a time-out feature in your interrupt handler. The MDHA will stall with no response if it is waiting for message data. This will most likely occur if the MDDSR write is not received or auto-padding is disabled and a partial message block is provided.

11. If MDSR[DONE] is set or done interrupt is triggered, then read the message digest from the message digest registers.

## 28.4.5 Performing an NMAC

An NMAC consists of one Hash operation.

1. Reset the MDHA using the MDCMR[SWR] bit.
2. MDCR register write. Enable the interrupts. (optional)
3. MDMR register write. Select algorithm, data padding, HMAC.
4. Direct context load of the second key into DDx1 registers from OPAD step.
5. MDMDS register write. Load this register with the length of the key.
6. Direct context load of the first key into MDx0 registers from IPAD step.
7. MDDSR register write. Load this register with the length of the message data (without padding) in bits.
8. Direct context load of MDMDS register with key size (for NMAC load with data size of 0 bytes).
9. Fill data FIFO with message to be hashed.
10. MDHA does the required algorithm's auto padding of the message.
11. Set the MDCMR[GO] bit.
12. Wait for MDSR[INT] to be set or done interrupt to be triggered to indicate successful completion (or failure).

## NOTE

You will need to provide a time-out feature in your interrupt handler. The MDHA will stall with no response if it is waiting for message data. This will most likely occur if the MDDSR write is not received or auto-padding is disabled and a partial message block is provided.

13. If MDSR[DONE] is set or done interrupt is triggered, then read the message digest from the message digest registers.

# Chapter 29

## Symmetric Key Hardware Accelerator (SKHA)

### 29.1 Introduction

The Symmetric Key Hardware Accelerator (SKHA) is a cryptographic hardware coprocessor designed to implement two widely used symmetric key block cipher algorithms, AES (Advanced Encryption Standard) and DES (Data Encryption Standard).

#### 29.1.1 Features

The SKHA supports the following block ciphers:

- AES
  - 128-bit key
  - Electronic Code Book (ECB), Cipher Block Chaining (CBC), Counter (CTR) cipher modes.
- DES
  - 64 bit key (with parity)
  - ECB, CBC, and CTR modes
- Triple-DES (3DES)
  - 2 key & 3 key (128-bits & 192-bits with parity)
  - Key parity check
  - ECB, CBC, and CTR modes

#### 29.1.2 Modes of Operation

The SKHA module can encrypt and decrypt the following security algorithms and cipher modes:

- Algorithms: AES, DES, or 3DES
- Cipher modes: ECB, CBC, CTR

##### 29.1.2.1 Data Encryption Standard (DES & 3DES) Algorithm

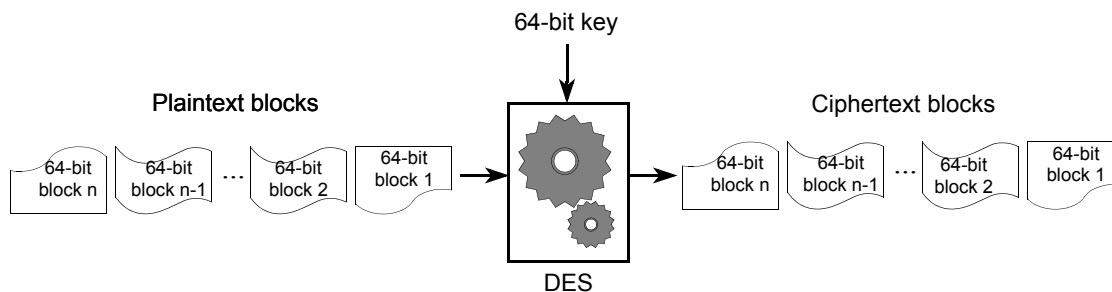
The SKHA is able to perform bulk data encryption/decryption in compliance with the Data Encryption Standard algorithm (ANSI x3.92, FIPS 46-2), as well as 3DES, which is an extension of the DES algorithm.

In DES mode, the SKHA operates by permuting 64-bit data blocks with a shared key and an initialization vector (IV). The SKHA supports two modes of IV operation: Electronic Code Book (ECB) and Cipher Block Chaining (CBC).

The processor supplies data to the SKHA, and the data will be encrypted and subsequently made available to the processor via an output FIFO. The session key is input to the block prior to encryption.

DES is a block cipher that uses a 56-bit key (64 bits with CRC) to encrypt 64-bit blocks of data, one block at a time. A conceptual diagram of this process is shown in [Figure 29-1](#). DES is a symmetric algorithm, so each of the two communicating parties share the same 64-bit key for encryption and decryption. DES processing begins after this shared session key is agreed upon. The text or binary message to be encrypted (typically called plaintext) is partitioned into  $n$  sets of 64-bit blocks. Each block is processed, in turn, by the DES engine, producing  $n$  sets of encrypted (ciphertext) blocks. These blocks may be transmitted to the other entity.

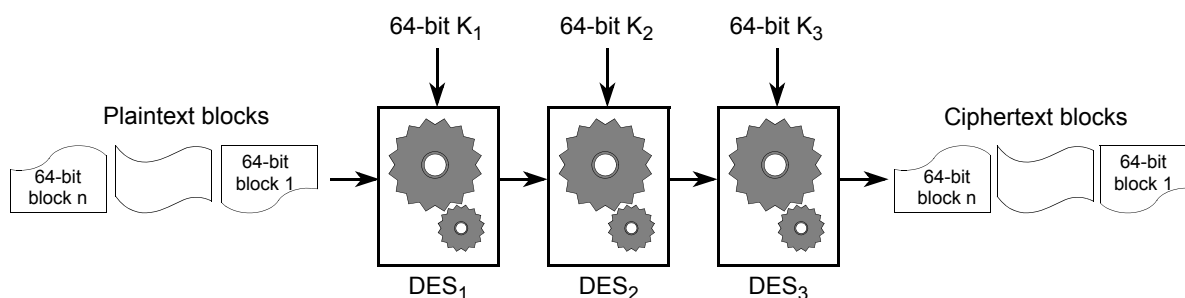
Decryption is handled in the reverse manner. The ciphertext blocks are processed one at a time by a DES module in the recipient's system. The same key is used, and the DES block manages the key processing internally so that the plaintext blocks are recovered.



**Figure 29-1. DES Encryption Process**

In addition, the SKHA module can compute 3DES, which is an extension to the DES algorithm whereby every 64-bit input block is processed three times. The SKHA supports two key ( $K1=K3$ ) or three key 3DES. A diagram of 3DES is shown in [Figure 29-2](#).



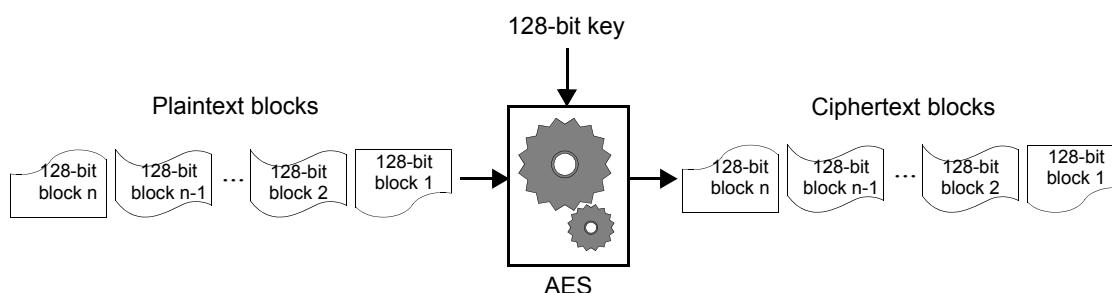


**Figure 29-2. Triple-DES Encryption Process (ECB Mode)**

### 29.1.2.2 Advanced Encryption Standard (AES) Algorithm

In AES mode, SKHA is used to accelerate bulk data encryption/decryption in compliance with the advanced encryption standard algorithm (AES) Rijndael. AES executes on 128-bit blocks with 128-bit key size.

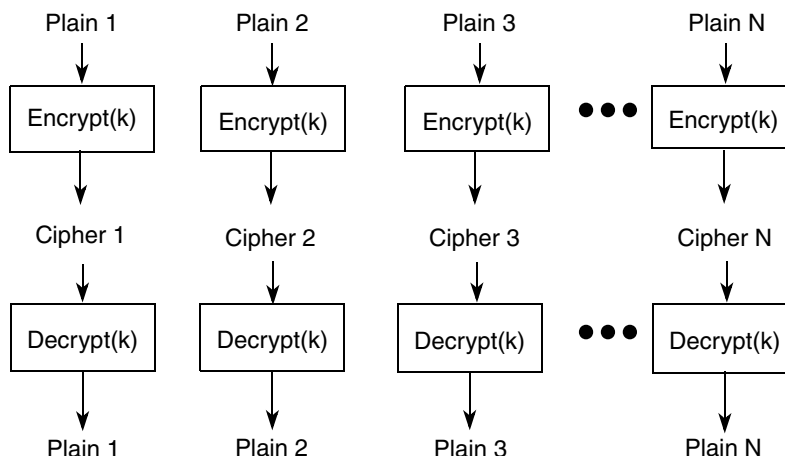
AES is a symmetric key algorithm, the sender and receiver use the same key for both encryption and decryption. The session key and initialization vector (CBC mode) are supplied to the SKHA module prior to encryption. The processor supplies data to the module that is processed as 128-bit input. The SKHA engine performs ten rounds for encryption or decryption. AES operates in ECB, CBC, and CTR modes.



**Figure 29-3. AES Encryption Process**

### 29.1.2.3 Electronic Code Book (ECB) Cipher Mode

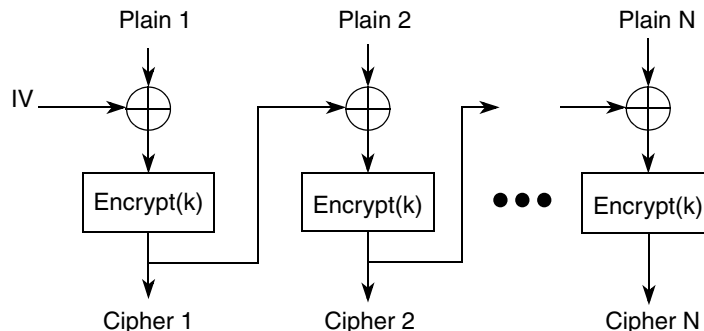
The simplest mode is ECB. Each block is processed independently, as shown in [Figure 29-4](#). There is no context used in this mode. Each block of plaintext is encrypted to determine the corresponding ciphertext. In decrypt mode, the ciphertext blocks are decrypted to restore the plaintext.



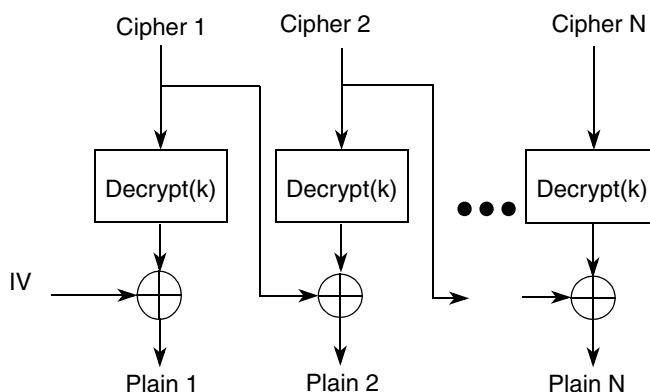
**Figure 29-4. ECB Mode Processing**

#### 29.1.2.4 Cipher Block Chaining (CBC) Cipher Mode

Cipher Block Chaining mode encrypts the output of the previous block with the current block input as shown in Figure 29-5. For decryption, the previous ciphertext block is mixed with the decrypted block as shown in Figure 29-6. A 128-bit random initialization vector (IV) must be loaded prior to processing a message. The context registers are updated internally and must be read and restored during a context switch.



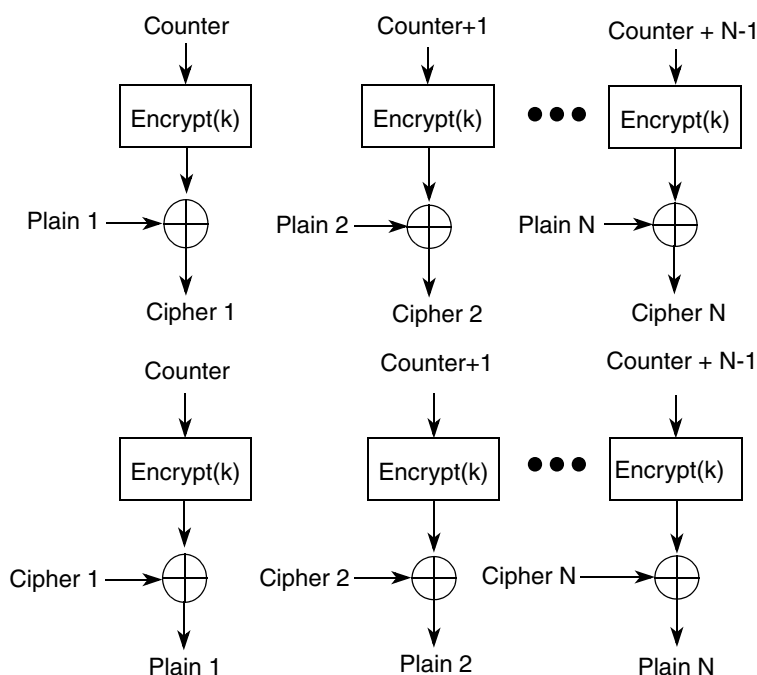
**Figure 29-5. CBC Encryption**



**Figure 29-6. CBC Decryption**

### 29.1.2.5 Counter (CTR) Cipher Mode

In counter mode, a random 128-bit initial counter value is incremented modulo  $2^n$  with each block processed. The modulus size can be set between  $2^8$  through  $2^{128}$ , by powers of 8 (SKMR[CTRM]). The running counter is encrypted and XOR'd with the plaintext to derive the ciphertext, or with the ciphertext to recover the plaintext as shown in [Figure 29-7](#).



**Figure 29-7. CTR Mode Processing**

## 29.2 Memory Map/Register Definition

Table 29-1 shows the SKHA module memory map.

**Table 29-1. Module Memory Map**

IPSBAR Offset	Mnemonic	[31:24]	[23:16]	[15:8]	[7:0]	Access
0x1B_0000	SKMR	SKHA Mode Register				R/W
0x1B_0004	SKCR	SKHA Control Register				R/W
0x1B_0008	SKCMR	SKHA Command Register				R/W
0x1B_000C	SKSR	SKHA Status Register				R
0x1B_0010	SKESR	SKHA Error Status Register				R
0x1B_0014	SKEMR	SKHA Error Status Mask Register				R/W
0x1B_0018	SKKSR	SKHA Key Size Register				R/W
0x1B_001C	SKDSR	SKHA Data Size Register				R/W
0x1B_0020	SKIN	SKHA Input FIFO				R/W
0x1B_0024	SKOUT	SKHA Output FIFO				R/W
0x1B_0030	SKK1	SKHA Key 1				W
0x1B_0034	SKK2	SKHA Key 2				W
0x1B_0038	SKK3	SKHA Key 3				W
0x1B_003C	SKK4	SKHA Key 4				W
0x1B_0040	SKK5	SKHA Key 5				W
0x1B_0044	SKK6	SKHA Key 6				W
0x1B_0070	SKC1	SKHA Context 1 (IV/Nonce/Running Offset/CTR/MAC)				R/W
0x1B_0074	SKC2	SKHA Context 2				R/W
0x1B_0078	SKC3	SKHA Context 3				R/W
0x1B_007C	SKC4	SKHA Context 4				R/W
0x1B_0080	SKC5	SKHA Context 5				R/W
0x1B_0084	SKC6	SKHA Context 6				R/W
0x1B_0088	SKC7	SKHA Context 7				R/W
0x1B_008C	SKC8	SKHA Context 8				R/W
0x1B_0090	SKC9	SKHA Context 9				R/W
0x1B_0094	SKC10	SKHA Context 10				R/W
0x1B_0098	SKC11	SKHA Context 11				R/W
0x1B_009C	SKC12	SKHA Context 12				R/W

## 29.2.1 Register Descriptions

### 29.2.1.1 SKHA Mode Register (SKMR)

The SKMR is used to select the cipher algorithm, direction, and cipher mode as shown in [Figure 29-8](#). If the mode is modified during message processing, the SKESR[RM] bit will be set. The mode may be modified once the module has completed processing, SKSR[DONE] is set.

#### NOTE

If reserved bits or undefined modes are set, a mode error will be generated.

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
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
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	CTRM				DKP	0	0	CM		DIR	ALG	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Address	IPSBAR + 0x1B_0000															

**Figure 29-8. SKHA Mode Register (SKMR)**

Table 29-2. SKMR Field Descriptions

Bits	Name	Description																																			
31–12	—	Reserved, should be cleared.																																			
11–8	CTRM	Counter mode modulus. Specifies modulus size for counter mode. In counter mode, the initial counter value will be incremented modulo $2^N$ . <table border="1"> <thead> <tr> <th rowspan="2">CTRM</th><th colspan="2">CTR Modulus</th></tr> <tr> <th>DES or 3DES</th><th>AES</th></tr> </thead> <tbody> <tr> <td>0000</td><td><math>2^8</math></td><td><math>2^8</math></td></tr> <tr> <td>0001</td><td><math>2^{16}</math></td><td><math>2^{16}</math></td></tr> <tr> <td>...</td><td>...</td><td>...</td></tr> <tr> <td>0110</td><td><math>2^{56}</math></td><td><math>2^{56}</math></td></tr> <tr> <td>0111</td><td><math>2^{64}</math></td><td><math>2^{64}</math></td></tr> <tr> <td>1000</td><td>Reserved<sup>1</sup></td><td><math>2^{72}</math></td></tr> <tr> <td>1001</td><td>Reserved<sup>1</sup></td><td><math>2^{80}</math></td></tr> <tr> <td>...</td><td>...</td><td>...</td></tr> <tr> <td>1110</td><td>Reserved<sup>1</sup></td><td><math>2^{120}</math></td></tr> <tr> <td>1111</td><td>Reserved<sup>1</sup></td><td><math>2^{128}</math></td></tr> </tbody> </table> <p><b>Note:</b> If CTRM is set to reserved settings for DES or 3DES, a mode error will occur.</p>	CTRM	CTR Modulus		DES or 3DES	AES	0000	$2^8$	$2^8$	0001	$2^{16}$	$2^{16}$	...	...	...	0110	$2^{56}$	$2^{56}$	0111	$2^{64}$	$2^{64}$	1000	Reserved <sup>1</sup>	$2^{72}$	1001	Reserved <sup>1</sup>	$2^{80}$	...	...	...	1110	Reserved <sup>1</sup>	$2^{120}$	1111	Reserved <sup>1</sup>	$2^{128}$
CTRM	CTR Modulus																																				
	DES or 3DES	AES																																			
0000	$2^8$	$2^8$																																			
0001	$2^{16}$	$2^{16}$																																			
...	...	...																																			
0110	$2^{56}$	$2^{56}$																																			
0111	$2^{64}$	$2^{64}$																																			
1000	Reserved <sup>1</sup>	$2^{72}$																																			
1001	Reserved <sup>1</sup>	$2^{80}$																																			
...	...	...																																			
1110	Reserved <sup>1</sup>	$2^{120}$																																			
1111	Reserved <sup>1</sup>	$2^{128}$																																			
7	DKP	Disable key parity check. Disables checking DES parity 0 Check for DES key parity errors 1 Do not check for DES key parity errors <b>Note:</b> A mode error will be generated if this bit is set to one while in AES mode.																																			
6–5	—	Reserved, should be cleared.																																			
4–3	CM	Cipher mode. Selects the cipher mode. 00 ECB 01 CBC 10 Reserved 11 CTR																																			
2	DIR	Direction. Selects encryption or decryption 0 Decrypt 1 Encrypt																																			
1–0	ALG	Algorithm. Selects which algorithm the SKHA module uses 00 AES 01 DES 10 3DES 11 Reserved																																			

### 29.2.1.2 SKHA Control Register (SKCR)

The SKCR contains bits that should be set after a hardware reset.

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
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
	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	IE
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Address	IPSBAR + 0x1B_0004															

Figure 29-9. SKHA Control Register (SKCR)

Table 29-3. SKCR Field Descriptions

Bits	Name	Description
31–1	—	Reserved
0	IE	Interrupt enable. 0 Interrupts disabled 1 Interrupts enabled

### 29.2.1.3 SKHA Command Register (SKCMR)

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
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
	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	GO	CI	RI	SWR
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Address	IPSBAR + 0x1B_0008															

Figure 29-10. SKHA Command Register (SKCMR)

Table 29-4. SKCMR Field Descriptions

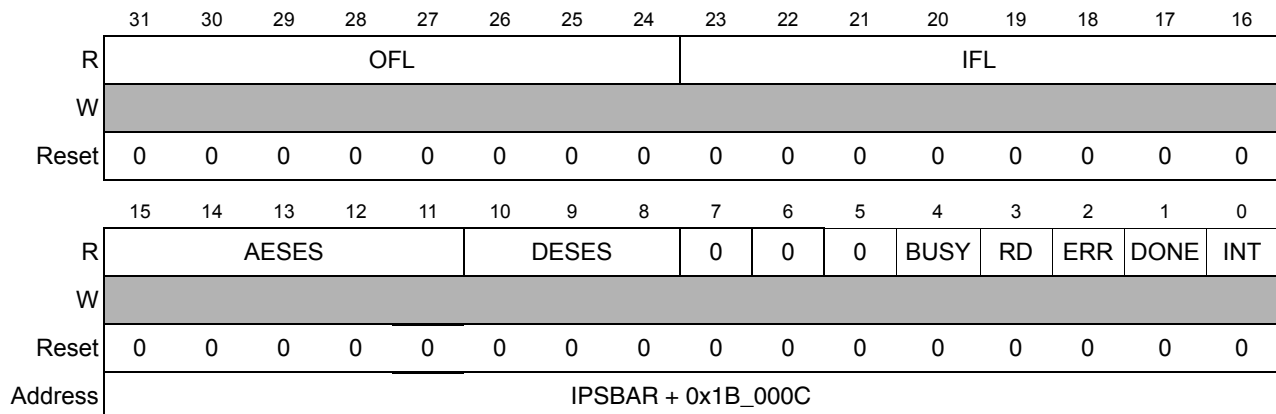
Bits	Name	Description
31–4	—	Reserved
3	GO	Go. Indicates that all data has been loaded into the module and the module should complete all processing. This bit is self resetting. 0 Do not finish processing 1 Complete all processing

**Table 29-4. SKCMR Field Descriptions (Continued)**

Bits	Name	Description
2	CI	Clear Interrupt Request. Clears errors in the SKHA error status registers and deasserts any pending interrupt request to the interrupt controller. This bit is self resetting. 0 Do not clear interrupts & errors 1 Clear interrupt requests & errors
1	RI	Reinitialize. Reinitializes memory and clears all registers except SKHA Error Status Mask and Control registers. This bit is self clearing. 0 No Reinitialization 1 Reinitialize SKHA module
0	SWR	Software Reset. Functionally equivalent to a hardware reset. All registers are reset and FIFOs are cleared. This bit is self clearing. 0 No reset 1 Perform software reset

### 29.2.1.4 SKHA Status Register (SKSR)

The SKSR is read-only and reflects the current state of the SKHA. It also contains the internal state values of the DES and AES state machines for the purposes of debugging. A write to this register has no effect.

**Figure 29-11. SKHA Status Register (SKSR)****Table 29-5. SKSR Field Descriptions**

Bits	Name	Description
31–24	OFL	Output FIFO level. This 8-bit value indicates the number of data words in the Output FIFO.
23–16	IFL	Input FIFO level. This 8-bit value indicates the number data words in the Input FIFO.
15–11	AESES	AES engine state. Current value of AES engine state machine (Debug only)
10–8	DESES	DES engine state. Current value of DES engine state machine (Debug only)
7–5	—	Reserved.



**Table 29-5. SKSR Field Descriptions (Continued)**

Bits	Name	Description
4	BUSY	Busy. Indicates the SKHA is busy. Mode, key data, context, and key size registers may not be modified and context registers may not be read while busy. 0 SKHA idle 1 SKHA busy
3	RD	Reset done. Indicates if reset of the SKHA module has completed. 0 Reset in progress 1 Reset complete
2	ERR	Error interrupt. Indicates that an error has occurred. 0 No error 1 Error occurred
1	DONE	Done interrupt. Indicates that the module has finished processing. 0 Not done 1 Done processing
0	INT	SKHA interrupt. Indicates that the module has finished processing data and the result is ready to be read from the Output FIFO or there is an error. This bit will assert when an interrupt request is generated unless the SKHACR[IE] bit is not set. 0 No interrupt 1 Done or error interrupt

### 29.2.1.5 SKHA Error Status Register (SKESR)

The read-only SKESR indicates the type of error that has occurred. These errors are described below and shown in [Table 29-6](#). When an error occurs, the SKHA engine will halt and assert an interrupt request to the interrupt controller. If multiple errors occur, only the first error will be flagged. The SKHA must be reset when any error occurs. A write to this register has no effect.

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
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
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	KRE	KPE	ERE	RMDP	KSE	DS E	IME	NEOF	NEIF	OFU	IFO
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Address	IPSBAR + 0x1B_0010															

**Figure 29-12. SKHA Error Status Register (SKESR)**

**Table 29-6. SKESR Field Descriptions**

Bits	Name	Bit Description
31–11	—	Reserved
10	KRE	Key read error. An illegal attempt to read the key registers during processing has been detected. 0 No error 1 Key read error occurred
9	KPE	Key parity error. Indicates if a DES key parity error has occurred. Key parity checking can be disabled by setting the SKMR[DKP] bit (See <a href="#">Section 29.2.1.1, “SKHA Mode Register (SKMR).”</a> ). 0 No error 1 Key parity error occurred
8	ERE	Early read. 0 No error 1 A context register was read from while the module was busy processing data.
7	RMDP	Register modified during processing 0 No error 1 A register was modified during processing
6	KSE	Key size Error 0 No error 1 Illegal key size was written into the SKHA key size register
5	DSE	Data size error 0 No error 1 Illegal data size was written into the SKHA data size register
4	IME	Illegal mode error. 0 No error 1 Illegal mode specified
3	NEOF	Non-empty output FIFO upon start. 0 No error 1 Output FIFO contains data upon start of processing
2	NEIF	Non-empty input FIFO upon done. 0 No error 1 Input FIFO contained data when processing was complete
1	OFU	Output FIFO underflow 0 No error 1 Output FIFO was read while empty
0	IFO	Input FIFO overflow. 0 No error 1 Input FIFO has been written to while full

### 29.2.1.6 SKHA Error Status Mask Register (SKESMR)

The SKESMR (located at address IPSBAR + 0x1B\_0014) allows the user to mask off any of the SKHA Error bits and not generate an interrupt. If the error occurs while the corresponding bit is set then the error will be set in the SKESR but the interrupt will not be generated. Additional errors will be flagged in the SKESR until an unmasked error is generated.

**NOTE**

Masking errors should only be used for debug purposes. A masked error will most likely cause invalid data.

**29.2.1.7 SKHA Key Size Register (SKKSR)**

The 6-bit SKKSR is used to set the number of bytes in the key. For AES, this value is always 16 (0x10). For Single DES, the key size must be 8 bytes (0x08). For two key Triple-DES the value should be 16 (0x10), while three key Triple-DES requires a key size of 24 bytes (0x18). The SKKSR must be set after the key data is written. Once the key size is set, the contents of the key register will be used to initialize the SKHA. If an illegal key size is specified, a key size error will occur, setting SKESR[KSE] and triggering an interrupt to the interrupt controller if KSE is not masked. If the key size is modified during message processing, a register modified error will be generated, setting SKESR[RMDP] and triggering an interrupt to the interrupt controller if RMDP is not masked.

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
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
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	—										Key Size in bytes					
W																
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Address	IPSBAR + 0x1B_0018															

**Figure 29-13. SKHA Key Size Register (SKKSR)****29.2.1.8 SKHA Data Size Register (SKDSR)**

The 32-bit SKDSR holds the number of bytes to process. This value is set prior to loading message data. The SKHA internally decrements this value as it processes the message. The SKDSR may be read at any time to determine the number of bytes that remain to be processed. This value may be modified during message processing. The value written will be internally added to the current value of data size. The SKHA will continue to process data until the value in SKDSR reaches zero.

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	Number of bytes to process															
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	Number of bytes to process															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Address	IPSBAR + 0x1B_001C															

**Figure 29-14. SKHA Data Size Register (SKDSR)**

The value of data size must be a multiple of 8 for DES/3DES or a multiple of 16 for AES. However, for CTR mode, data size may be a multiple of 8. If an illegal data size is set, a data size error will be generated, setting SKESR[DSE] and generating an interrupt request to the interrupt controller if the DSE bit is not masked.

### 29.2.1.9 SKHA Input FIFO

The SKHA Input FIFO provides temporary storage for data to be used during processing. The input to the FIFO is accessible through the address IPSBAR + 0x1B\_0020 in the address map. The input FIFO is a write-only register and attempting to read from this register will always return zero. If the FIFO is written to when the FIFO level is full then an interrupt will be generated and SKESR[IFO] bit will be set. The SKSR[IFL] field, described in [Section 29.2.1.4, “SKHA Status Register \(SKSR\),”](#) can be polled to monitor how many 32-bit words are currently resident in the FIFO.

### 29.2.1.10 SKHA Output FIFO

The SKHA Output FIFO provides temporary storage for data post-processing. The output from the FIFO is accessible through the address IPSBAR + 0x1B\_0024 in the address map. The output FIFO is a read-only register and attempting to write to this register has no effect. If the output FIFO is read from when the FIFO level is empty then an interrupt is generated and SKESR[OFU] is set. The SKSR[OFL] field, described in [Section 29.2.1.4, “SKHA Status Register \(SKSR\),”](#) can be polled to monitor how many 32-bit words are currently resident in the FIFO.

### 29.2.1.11 SKHA Key Data Registers (SKKDR<sub>n</sub>)

The six 32-bit SKKDR<sub>n</sub> are write-only and hold the symmetric key. The key should be loaded with the first four bytes placed in Key 1, followed by Key 2, etc. The key should be loaded before setting the key size. Any key data beyond the value specified in the SKKSR will be ignored.

For AES, which uses a 128-bit key, SKKDR1–SKKDR4 must be written. For single DES, SKKDR1–SKKDR2 must be written (56-bit key & 8-bit parity). For 2-key 3DES, SKKDR1–4 must be written. For 3-key 3DES, SKKDR1–6 must be written. The SKKDR $n$  registers are summarized in [Table 29-7](#).

**Table 29-7. SKHA Key Data Register Definitions**

Algorithm	SKKDR1	SKKDR2	SKKDR3	SKKDR4	SKKDR5	SKKDR6
AES	Bytes 1–4	Bytes 5–8	Bytes 9–12	Bytes 13–16	—	—
DES	Bytes 1–4	Bytes 5–8	—	—	—	—
2 key 3DES	K1 Bytes 1–4	K1 Bytes 5–8	K2 Bytes 1–4	K2 Bytes 5–8	—	—
3 key 3DES	K1 Bytes 1–4	K1 Bytes 5–8	K2 Bytes 1–4	K2 Bytes 5–8	K3 Bytes 1–4	K3 Bytes 5–8

Attempting to read the SKKDR $n$  registers will generate an illegal address error. If the key registers are modified during message processing, a register modify error will be generated, setting SKESR[RMDP] and triggering an interrupt to the interrupt controller (if RMDP is not masked).

### 29.2.1.12 SKHA Context Registers (SKCR $n$ )

Prior to loading key data, the user must write the appropriate initialization data to the SKHA. Following a done interrupt, the user must read the contents of the SKCR $n$  registers prior to switching context. These values must be restored to resume processing of the original message. The SKCR $n$  registers are loaded differently depending on the algorithm and cipher mode. The location and values are summarized in [Table 29-8](#). Context should be loaded with the lower bytes in the lowest 32-bit context register.

Ex: 0x0123456789ABCDEF101112131415161718191A1B1C1D1E1F would be loaded as follows:

SKCR1 = 0x0102030405060708090A0B0C0D0E0F

SKCR2 = 0x101112131415161718191A1B1C1D1E1F

**Table 29-8. SKHA Context Register Definitions**

	SKHA Context Register $n$ (32-bits each)											
Algorithm /Mode	1	2	3	4	5	6	7	8	9	10	11	12
DES-ECB	—	—	—	—	—	—	—	—	—	—	—	—
DES-CBC	IV*		—	—	—	—	—	—	—	—	—	—
AES-ECB	—	—	—	—	—	—	—	—	—	—	—	—
AES-CBC	IV*				—	—	—	—	—	—	—	—
DES-CTR	Counter*		—	—	—	—	—	—	—	—	—	—

**Table 29-8. SKHA Context Register Definitions (Continued)**

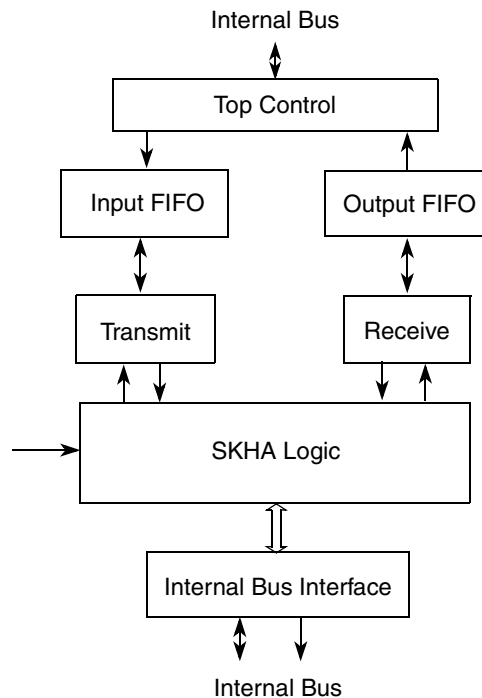
	SKHA Context Register <i>n</i> (32-bits each)											
Algorithm /Mode	1	2	3	4	5	6	7	8	9	10	11	12
AES-CTR	Counter*				—	—	—	—	—	—	—	—
*Must be written at start of new message												

The value written to the IV/Nonce registers will be replaced with the running initialization vector, IV, (CBC mode) or running counter (CTR mode) once processing begins.

If any context registers are read prior to the SKSR[DONE] bit being set, an early read error, SKESR[ERE], will be generated and an interrupt request will be sent to the interrupt controller (if ERE is not masked).

## 29.3 Functional Description

At the top level, the SKHA consists of seven functional blocks: The input FIFO, the output FIFO, transmit FIFO interface, receive FIFO interface, internal bus interface, top control and the SKHA logic.

**Figure 29-15. SKHA Block Diagram**

### 29.3.1 Transmit FIFO Interface Block

This block translates the internal FIFO control signals to the input FIFO and passes the message data to the SKHA logic block. The SKHA logic block will continue to pop data from the input FIFO until the FIFO is empty.

For AES, four 32-bit words are required to process a block. For DES/3DES, two 32-bit words are required to process a block.

### 29.3.2 Receive FIFO Interface Block

This block translates the internal FIFO control signals to the output FIFO and pass the processed message data from the SKHA logic block. The SKHA logic block will push the same number of words to the output FIFO as it pops from the input FIFO. The SKHA logic block will push to the output FIFO as long as the FIFO is not full. Once the last word is pushed to the output FIFO, the done interrupt will be asserted.

### 29.3.3 Top Control Block

This block generates the input and output FIFO transmit, receive and request signals and translates other internal signals at the top level.

### 29.3.4 SKHA Logic Block

This block contains the internal address decoder, addressable registers (key data, key size, data size, mode, context data, and status), interrupt and error logic, and the core engine as shown in Figure 29-16.

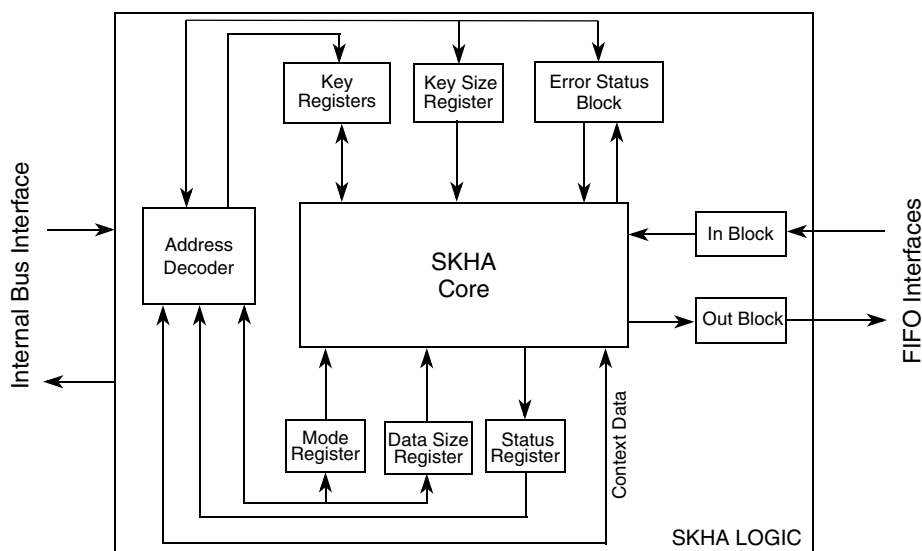


Figure 29-16. SKHA Logic Block Diagram

### 29.3.4.1 Address Decode Logic

The address decoder translates the internal address to write to the proper SKHA registers.

### 29.3.4.2 Error Interrupt/Status Logic

This block generates the error interrupt if the host performs an illegal operation. The cause of the error is flagged in the SKHA error status register ([Section 29.2.1.5, “SKHA Error Status Register \(SKESR\)”](#)) and an interrupt is triggered to the interrupt controller. If an error occurs, the SKHA core engine is halted. This prevents the core from continuing operation with invalid data. These error interrupts may be masked off selectively by setting the appropriate bits in the SKHA error status mask register ([Section 29.2.1.6, “SKHA Error Status Mask Register \(SKESMR\)”](#))

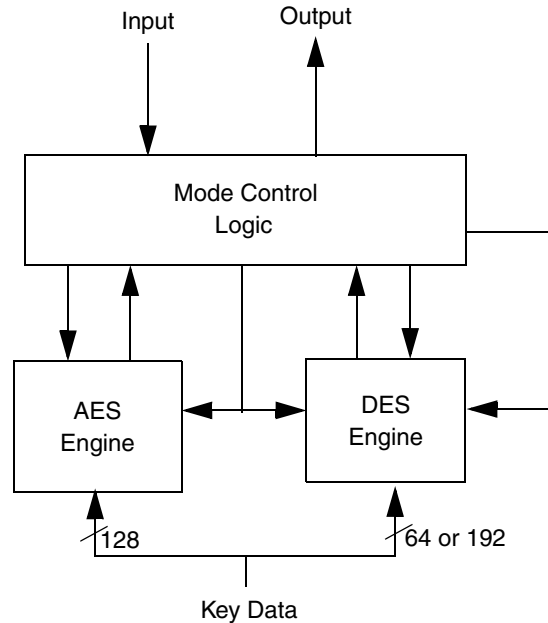
### 29.3.4.3 SKHA Core

The heart of the SKHA is the core processing engine, [Figure 29-17](#). The core contains the DES and AES block cipher engines. Also, the cipher mode (ECB, CBC, CTR) is implemented in this block. The SKHA logic block drives the cipher mode, algorithm, processing direction, key, and input block to the Mode Control logic.

While DES and AES operate differently internally, the Mode Control logic operates on top of the AES and DES engines. The Mode Control logic interfaces to both engines and feeds the input block and key to the selected engine. When the selected engine processes a block, it returns a "done" signal with the output block. The Mode Control logic in turn returns a "done" signal to the SKHA logic block along with the processed message block.

When the entire message is processed (following write to the “End of Message” register), the SKHA logic block sets SKSR[DONE] and generates an interrupt request to the interrupt controller. This will indicate to the user that it is safe to read context.





**Figure 29-17. SKHA Core Block Diagram**

### 29.3.5 Security Assurance Features

The SKHA features simple security assurance features to prevent operation in the presence of hardware faults and to shield visibility to sensitive data. Further, any user error or internal hardware fault will cause the internal state machines to enter an idle/error state. In this case, the SKHA must be reset to resume operation, SKCMR[SWR]. Readable registers may be accessed to determine the nature of the error.

## 29.4 Initialization/Application Information

### 29.4.1 General Operation

The intended general operation of the SKHA is as follows for ECB, CBC and CTR modes:

1. Reset/initialize.
2. Set algorithm, processing direction, and cipher mode in the SKMR.
3. Write initialization vector/counter to the SKCR $n$  registers, if necessary.
4. Write symmetric key to SKKDR $n$  registers.
5. Set number of bytes in key in the SKKSR.
6. Set the number of bits in the message in the SKDSR.
7. Load the input FIFO with message data.
8. Set the SKCMR[GO] bit.

9. Wait for interrupt, SKSR[INT].
10. Unload processed message data from the output FIFO.
11. Read contents of context registers, if necessary.
12. Set the SKCMR[CI] bit, to clear the done interrupt.

### 29.4.2 Operation with Context Switch

The intended operation to process part of one message, followed by an intermediate message, and resume processing the original message is as follows:

1. Reset/initialize.
2. Set algorithm, set processing direction and cipher mode in the SKMR.
3. Write IV/Nonce to SKCR $n$  registers.
4. Write the symmetric key to SKKDR $n$  registers.
5. Write the number of bytes in the key to SKKSR.
6. Set number of bits in the first part of message 1 in the SKDSR.
7. Load the input FIFO with first part of message 1.
8. Set the SKCMR[GO] bit.
9. Wait for done interrupt, SKSR[DONE].
10. Unload processed message data from the output FIFO.
11. Read context registers and store contents in memory.
12. Set the SKCMR[CI] bit, to clear the done interrupt.
13. Reset/initialize.
14. Process intermediate message as described in [Section 29.4.1, “General Operation.”](#)
15. Reset/initialize.
16. Set algorithm, processing direction and cipher mode in the SKMR.
17. Restore context registers from memory.
18. Restore symmetric key to SKKDR $n$ .
19. Write number of bytes in the key to the SKKSR.
20. Set number of bits in remaining part of message 1 to SKDSR.
21. Load the input FIFO with the remaining portion of message 1.
22. Set the SKCMR[GO] bit.
23. Wait for interrupt, SKSR[INT].
24. Unload processed message data from the output FIFO.

# Chapter 30

## Random Number Generator (RNG)

### 30.1 Introduction

This chapter describes the Random Number Generator (RNG), including a programming model, functional description, and application information.

#### 30.1.1 Overview

The Random Number Generator (RNG) module is capable of generating 32-bit random numbers. It is designed to comply with FIPS-140 standards for randomness and non-determinism. The random bits are generated by clocking shift registers with clocks derived from ring oscillators. The configuration of the shift registers ensures statistically good data (i.e. data that looks random). The oscillators with their unknown frequencies provide the required entropy needed to create random data.

### 30.2 Memory Map/Register Definition

The address map for the RNG module is shown in [Table 30-1](#). Detailed register descriptions are found in the following section.

**Table 30-1. RNG Module Memory Map**

IPSBAR Offset	Mnemonic	[31:24]	[23:16]	[15:8]	[7:0]	Access
0x1A_0000	RNGCR	RNG Control Register				R/W
0x1A_0004	RNGSR	RNG Status Register				R
0x1A_0008	RNGER	RNG Entropy Register				W
0x1A_000C	RNGOUT	RNG Output FIFO				R

#### 30.2.1 RNG Control Register (RNGCR)

Immediately following reset, the RNG begins generating entropy in its internal shift registers. Random data is not pushed to the output FIFO until after the GO bit in the RNGCR is set to a one. After this, a random 32-bit word is pushed to the FIFO every 256 cycles. If the FIFO is full, then no push will occur.

In this way the FIFO will be kept as close to full as possible. The fields in the RNGCR are defined in [Table 30-2](#).

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
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
	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	CI	IM	HA	GO
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Address	IPSBAR + 0x1A_0000															

Figure 30-1. RNG Control Register (RNGCR)

Table 30-2. RNGCR Field Descriptions

Bits	Name	Description
31–4	—	Reserved
3	CI	Clear interrupt. Writing a one to this bit clears the error interrupt and RNGSR[EI]. 0 Do not clear error interrupt. 1 Clear error interrupt.
2	IM	Interrupt mask. 0 Error interrupt is enabled. 1 Error interrupt is masked.
1	HA	High assurance. Notifies core when FIFO underflow has occurred (FIFO is read while empty). Enables the Security Violation bit in the RNGSR. Bit is sticky and can only be cleared by hardware reset. 0 Disable security violation notification. 1 Enable security violation notification.
0	GO	Go bit. Starts/stops random data from being generated. Bit is sticky and can only be cleared by hardware reset. 0 FIFO is not loaded with random data. 1 FIFO will be loaded with random data.

### 30.2.2 RNG Status Register (RNGSR)

The RNGSR, shown in [Figure 30-2](#), is a read only register which reflects the internal status of the RNG.

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	OFS							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	OFL								0	0	0	0	EI	FUF	LRS	SV
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Address	IPSBAR + 0x1A_0004															

**Figure 30-2. RNG Status Register (RNGSR)****Table 30-3. RNGSR Field Descriptions**

Bits	Name	Description
31–24	—	Reserved
23–16	OFS	Output FIFO size. Indicates size of the Output FIFO (16 words) & maximum possible value of RNGR[OFL].
15–8	OFL	Output FIFO level. Indicates current number of random words in the Output FIFO. Used to determine if valid random data is available for reading from the FIFO without causing an underflow condition.
7–4	—	Reserved
3	EI	Error interrupt. Signals a FIFO underflow. Reset by a write to RNGCR[CI] and not masked by RNGCR[IM]. 0 FIFO not read while empty. 1 FIFO read while empty.
2	FUF	FIFO underflow. Signals FIFO underflow. Reset by reading status register. 0 FIFO not read while empty, since last read of RNGSR. 1 FIFO read while empty, since last read of RNGSR.
1	LRS	Last read status. Reflects status of most recent read of the FIFO 0 During last read, FIFO was not empty. 1 During last read, FIFO was empty (underflow condition).
0	SV	Security violation. When enabled by RNGCR[HA], signals that a FIFO underflow has occurred. Bit is sticky and is only cleared by hardware reset 0 No violation occurred or RNGCR[HA] is cleared. 1 Security violation (FIFO underflow) has occurred.

### 30.2.3 RNG Entropy Register (RNGER)

The RNGER is a write-only register which allows the user to insert entropy into the RNG. This register allows an external user to continually seed the RNG with externally generated random data. Although the use of this register is recommended, it is optional. The RNGER can be written at any time during operation.

Each time the RNGER is written, the value is used to update the internal state of the RNG. The update is performed in such a way that the entropy in the RNG's internal state is preserved. Use of the RNGER can increase the entropy but never decrease it.

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	ENT															
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	ENT															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Address	IPSBAR + 0x1A_0008															

**Figure 30-3. RNG Entropy Register (RNGER)**

### 30.2.4 RNG Output FIFO (RNGOUT)

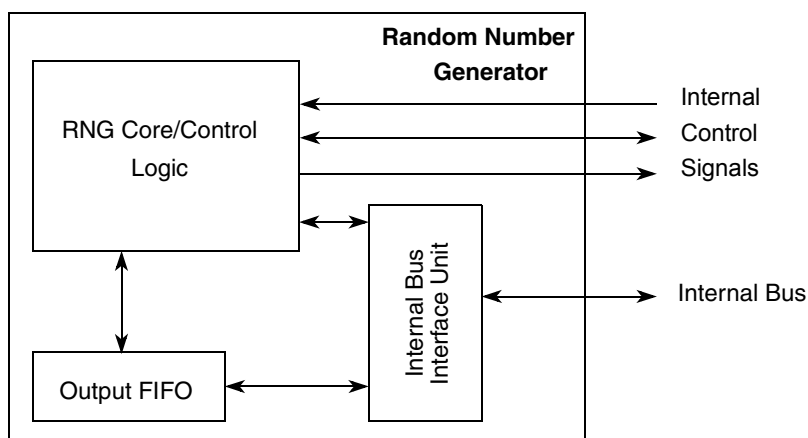
The RNGOUT provides temporary storage for random data generated by the RNG. As long as the FIFO is not empty, a read of this address will return 32 bits of random data. If the FIFO is read when it is empty, Error Interrupt, FIFO Underflow and Last Read bits in the RNGSR will be set. If the interrupt is enabled in the RNGCR an interrupt will be triggered to the interrupt controller. The RNGSR[OFL], described in [Section 30.2.2, “RNG Status Register \(RNGSR\)”](#), can be polled to monitor how many 32-bit words are currently resident in the FIFO. A new random word is pushed into the FIFO every 256 clock cycles (as long as the FIFO is not full). It is very important that the user polls RNGSR[OFL] to make sure random values are present before reading from the FIFO.

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	Random Output															
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	Random Output															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Address	IPSBAR + 0x1A_000C															

**Figure 30-4. RNG Output FIFO (RNGOUT)**

## 30.3 Functional Description

The RNG has three functional blocks. They are the Output FIFO, internal bus interface and the RNG Core/Control Logic blocks, as shown in Figure 30-5. The following sections describe these blocks in more detail.



**Figure 30-5. RNG Block Diagram**

### 30.3.1 Output FIFO

The Output FIFO provides temporary storage for random data generated by the RNG Core/Control Logic block. This allows the user to read multiple random long words back to back. The RNGSR, allows the user to monitor the number of random words in the FIFO through the Output FIFO Level field. If the user reads from the FIFO when it is empty and the interrupt is enabled, the RNG will drive an interrupt request to the interrupt controller. It is very important that the user polls the RNGSR[OFL] bit to make sure random values are present before reading from the FIFO.

### 30.3.2 RNG Core/Control Logic Block

This block contains the RNG's control logic as well as its core engine used to generate random data.

### 30.3.3 RNG Control Block

The Control Block contains the address decoder, all addressable registers, and control state machines for the RNG. This block is responsible for communication with both the peripheral interface and the FIFO interface. The block also controls the Core Engine to generate random data. The general functionality of the block is as follows. After reset, entropy is generated and stored in the RNG's shift registers. After the GO Bit is set in the RNGCR, the FIFO is loaded with a random word every 256 cycles. The process of loading the FIFO continues as long as the FIFO is not full.

### 30.3.4 RNG Core Engine

The Core Engine Block contains the logic used to generate random data. The logic within the Core Engine contains the internal shift registers as well as the logic used to generate the two oscillator based clocks. This logic is "brainless" and must be controlled by the Control Block. The Control Block controls how the shift registers are configured as well as when the oscillator clocks are turned on.

## 30.4 Initialization/Application Information

The intended general operation of the RNG is as follows:

1. Reset/initialize
2. Write to the RNG entropy register (optional).
3. Write to the RNG control register and set the interrupt mask, high assurance, and GO bits.
4. Poll RNG status register for FIFO level
5. Read available random data from RNG output FIFO
6. Repeat steps 3 and 4 as needed



# Chapter 31

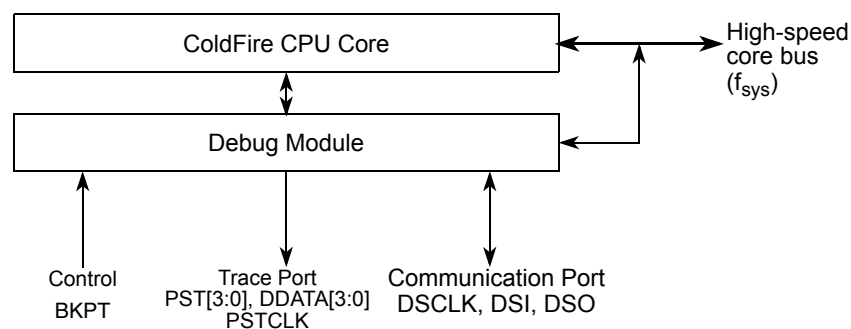
## Debug Support

### 31.1 Introduction

This chapter describes the Revision A enhanced hardware debug support in the MCF5275.

#### 31.1.1 Overview

The debug module is shown in [Figure 31-1](#).



**Figure 31-1. Processor/Debug Module Interface**

Debug support is divided into three areas:

- Real-time trace support—The ability to determine the dynamic execution path through an application is fundamental for debugging. The ColdFire solution implements an 8-bit parallel output bus that reports processor execution status and data to an external emulator system. See [Section 31.3, “Real-Time Trace Support.”](#)
- Background debug mode (BDM)—Provides low-level debugging in the ColdFire processor complex. In BDM, the processor complex is halted and a variety of commands can be sent to the processor to access memory and registers. The external emulator uses a three-pin, serial, full-duplex channel. See [Section 31.5, “Background Debug Mode \(BDM\),”](#) and [Section 31.4, “Memory Map/Register Definition.”](#)
- Real-time debug support—BDM requires the processor to be halted, which many real-time embedded applications cannot do. Debug interrupts let real-time systems execute a unique service routine that can quickly save the contents of key registers and variables and return the system to normal operation. External development systems can access saved data because the hardware supports concurrent operation of the processor and BDM-initiated commands. See [Section 31.6, “Real-Time Debug Support.”](#)

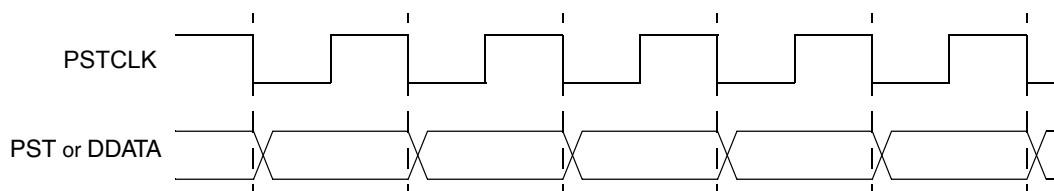
## 31.2 External Signal Description

Table 31-1 describes debug module signals. All ColdFire debug signals are unidirectional and related to a rising edge of the processor's clock signal. The standard 26-pin debug connector is shown in Section 31.8, "Freescale-Recommended BDM Pinout."

**Table 31-1. Debug Module Signals**

Signal	Description
Development Serial Clock (DSCLK)	Internally synchronized input. (The logic level on DSCLK is validated if it has the same value on two consecutive rising PSTCLK edges.) Clocks the serial communication port to the debug module during packet transfers. Maximum frequency is 1/5 the processor status clock (PSTCLK) speed. At the synchronized rising edge of DSCLK, the data input on DSI is sampled and DSO changes state.
Development Serial Input (DSI)	Internally synchronized input that provides data input for the serial communication port to the debug module.
Development Serial Output (DSO)	Provides serial output communication for debug module responses. DSO is registered internally.
Breakpoint ( $\overline{\text{BKPT}}$ )	Input used to request a manual breakpoint. Assertion of $\overline{\text{BKPT}}$ puts the processor into a halted state after the current instruction completes. Halt status is reflected on processor status signals (PST[3:0]) as the value 0xF.
PSTCLK	See Figure 31-2. PSTCLK indicates when the development system should sample PST and DDATA values.
Debug Data (DDATA[3:0])	These output signals display the register breakpoint status as a default, or optionally, captured address and operand values. The capturing of data values is controlled by the setting of the CSR. Additionally, execution of the WDDATA instruction by the processor captures operands which are displayed on DDATA. These signals are updated each processor cycle.
Processor Status (PST[3:0])	These output signals report the processor status. Table 31-2 shows the encoding of these signals. These outputs indicate the current status of the processor pipeline and, as a result, are not related to the current bus transfer. The PST value is updated each processor cycle.

Figure 31-2 shows PSTCLK timing with respect to PST and DDATA.



**Figure 31-2. PSTCLK Timing**

## 31.3 Real-Time Trace Support

Real-time trace, which defines the dynamic execution path, is a fundamental debug function. The ColdFire solution is to include a parallel output port providing encoded processor status and data to an external development system. This port is partitioned into two 4-bit nibbles: one nibble allows the processor to transmit processor status, (PST), and the other allows operand data to be

displayed (debug data, DDATA). The processor status may not be related to the current bus transfer.

External development systems can use PST outputs with an external image of the program to completely track the dynamic execution path. This tracking is complicated by any change in flow, especially when branch target address calculation is based on the contents of a program-visible register (variant addressing). DDATA outputs can be configured to display the target address of such instructions in sequential nibble increments across multiple processor clock cycles, as described in [Section 31.3.1, “Begin Execution of Taken Branch \(PST = 0x5\).”](#) Two 32-bit storage elements form a FIFO buffer connecting the processor’s high-speed local bus to the external development system through PST[3:0] and DDATA[3:0]. The buffer captures branch target addresses and certain data values for eventual display on the DDATA port, one nibble at a time starting with the least significant bit (lsb).

Execution speed is affected only when both storage elements contain valid data to be dumped to the DDATA port. The core stalls until one FIFO entry is available.

[Table 31-2](#) shows the encoding of these signals.

**Table 31-2. Processor Status Encoding**

PST[3:0]		Definition
Hex	Binary	
0x0	0000	Continue execution. Many instructions execute in one processor cycle. If an instruction requires more processor clock cycles, subsequent clock cycles are indicated by driving PST outputs with this encoding.
0x1	0001	Begin execution of one instruction. For most instructions, this encoding signals the first processor clock cycle of an instruction’s execution. Certain change-of-flow opcodes, plus the PULSE and WDDATA instructions, generate different encodings.
0x2	0010	Reserved
0x3	0011	Entry into user-mode. Signaled after execution of the instruction that caused the ColdFire processor to enter user mode.
0x4	0100	Begin execution of PULSE and WDDATA instructions. PULSE defines logic analyzer triggers for debug and/or performance analysis. WDDATA lets the core write any operand (byte, word, or longword) directly to the DDATA port, independent of debug module configuration. When WDDATA is executed, a value of 0x4 is signaled on the PST port, followed by the appropriate marker, and then the data transfer on the DDATA port. Transfer length depends on the WDDATA operand size.
0x5	0101	Begin execution of taken branch. For some opcodes, a branch target address may be displayed on DDATA depending on the CSR settings. CSR also controls the number of address bytes displayed, indicated by the PST marker value preceding the DDATA nibble that begins the data output. See <a href="#">Section 31.3.1, “Begin Execution of Taken Branch (PST = 0x5).”</a>
0x6	0110	Reserved
0x7	0111	Begin execution of return from exception (RTE) instruction.

**Table 31-2. Processor Status Encoding (Continued)**

PST[3:0]		Definition
Hex	Binary	
0x8– 0xB	1000– 1011	Indicates the number of bytes to be displayed on the DDATA port on subsequent processor clock cycles. The value is driven onto the PST port one PSTCLK cycle before the data is displayed on DDATA. 0x8 Begin 1-byte transfer on DDATA. 0x9 Begin 2-byte transfer on DDATA. 0xA Begin 3-byte transfer on DDATA. 0xB Begin 4-byte transfer on DDATA.
0xC	1100	Exception processing. Exceptions that enter emulation mode (debug interrupt or optionally trace) generate a different encoding, as described below. Because the 0xC encoding defines a multiple-cycle mode, PST outputs are driven with 0xC until exception processing completes.
0xD	1101	Entry into emulator mode. Displayed during emulation mode (debug interrupt or optionally trace). Because this encoding defines a multiple-cycle mode, PST outputs are driven with 0xD until exception processing completes.
0xE	1110	Processor is stopped. Appears in multiple-cycle format when the MCF5275 executes a STOP instruction. The ColdFire processor remains stopped until an interrupt occurs, thus PST outputs display 0xE until the stopped mode is exited.
0xF	1111	Processor is halted. Because this encoding defines a multiple-cycle mode, the PST outputs display 0xF until the processor is restarted or reset. (see <a href="#">Section 31.5.1, “CPU Halt”</a> )

### 31.3.1 Begin Execution of Taken Branch (PST = 0x5)

PST is 0x5 when a taken branch is executed. For some opcodes, a branch target address may be displayed on DDATA depending on the CSR settings. CSR also controls the number of address bytes displayed, which is indicated by the PST marker value immediately preceding the DDATA nibble that begins the data output.

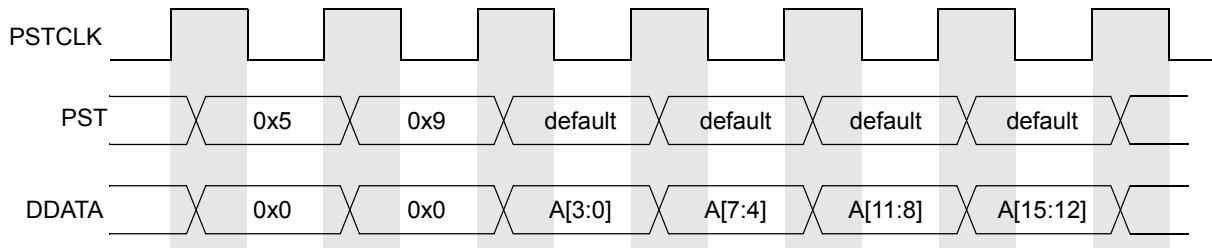
Bytes are displayed in least-to-most-significant order. The processor captures only those target addresses associated with taken branches which use a variant addressing mode, that is, RTE and RTS instructions, JMP and JSR instructions using address register indirect or indexed addressing modes, and all exception vectors.

The simplest example of a branch instruction using a variant address is the compiled code for a C language case statement. Typically, the evaluation of this statement uses the variable of an expression as an index into a table of offsets, where each offset points to a unique case within the structure. For such change-of-flow operations, the MCF5275 uses the debug pins to output the following sequence of information on successive processor clock cycles:

1. Use PST (0x5) to identify that a taken branch was executed.
2. Using the PST pins, optionally signal the target address to be displayed sequentially on the DDATA pins. Encodings 0x9–0xB identify the number of bytes displayed.

3. The new target address is optionally available on subsequent cycles using the DDATA port. The number of bytes of the target address displayed on this port is configurable (2, 3, or 4 bytes).

Another example of a variant branch instruction would be a JMP (A0) instruction. [Figure 31-3](#) shows the PST and DDATA outputs that indicate a JMP (A0) execution (assuming the CSR was programmed to display the lower 2 bytes of an address).



**Figure 31-3. Example JMP Instruction Output on PST/DDATA**

PST 0x5 indicates a taken branch and the marker value 0x9 indicates a 2-byte address. Thus, the subsequent 4 nibbles of DDATA display the lower 2 bytes of address register A0 in least-to-most-significant nibble order. The PST output after the JMP instruction completes depends on the target instruction. The PST can continue with the next instruction before the address has completely displayed on DDATA because of the DDATA FIFO. If the FIFO is full and the next instruction has captured values to display on DDATA, the pipeline stalls (PST = 0x0) until space is available in the FIFO.

## 31.4 Memory Map/Register Definition

In addition to the existing BDM commands that provide access to the processor's registers and the memory subsystem, the debug module contains 19 registers to support the required functionality. These registers are also accessible from the processor's supervisor programming model by executing the WDEBUG instruction (write only). Thus, the breakpoint hardware in the debug module can be written by the external development system using the debug serial interface or by the operating system running on the processor core. Software is responsible for guaranteeing that accesses to these resources are serialized and logically consistent. Hardware provides a locking mechanism in the CSR to allow the external development system to disable any attempted writes by the processor to the breakpoint registers (setting CSR[IPW]). BDM commands must not be issued if the MCF5275 is using the WDEBUG instruction to access debug module registers or the resulting behavior is undefined.

These registers, shown in [Figure 31-4](#), are treated as 32-bit quantities, regardless of the number of implemented bits.

**Table 31-3. Debug Programming Model**

[31:24]	[23:16]	[15:8]	[7:0]	Mnemonic
—		Address Attribute Trigger Register		AATR
Address Low Breakpoint Register				ABLR
Address High Breakpoint Register				ABHR
Configuration/Status Register				CSR
Data Breakpoint Register				DBR
Data Breakpoint Mask Register				DBMR
PC Breakpoint Register				PBR
PC Breakpoint Mask Register				PBMR
Trigger Definition Registers				TDR

**Note:** Each debug register is accessed as a 32-bit register; reserved fields above are not used (don't care). All debug control registers are writable from the external development system or the CPU via the WDEBUG instruction. CSR is write-only from the programming model. It can be read or written through the BDM port using the rdmreg and wdmreg commands.

**Figure 31-4. Debug Programming Model**

These registers are accessed through the BDM port by the commands, WDMREG and RDMREG, described in [Section 31.5.3.3, “Command Set Descriptions.”](#) These commands contain a 5-bit field, DRc, that specifies the register, as shown in [Table 31-4.](#)

**Table 31-4. BDM/Breakpoint Registers**

DRc[4–0]	Register Name	Mnemonic	Initial State	Page
0x00	Configuration/status register	CSR	0x0001_0000	p. 31-9
0x01–0x05	Reserved	—	—	—
0x06	Address attribute trigger register	AATR	0x0000_0005	p. 31-7
0x07	Trigger definition register	TDR	0x0000_0000	p. 31-14
0x08	Program counter breakpoint register	PBR	—	p. 31-13
0x09	Program counter breakpoint mask register	PBMR	—	p. 31-13
0x0A–0x0B	Reserved	—	—	—
0x0C	Address breakpoint high register	ABHR	—	p. 31-8
0x0D	Address breakpoint low register	ABLR	—	p. 31-8
0x0E	Data breakpoint register	DBR	—	p. 31-12
0x0F	Data breakpoint mask register	DBMR	—	p. 31-12

**NOTE**

Debug control registers can be written by the external development system or the CPU through the WDEBUG instruction.

CSR is write-only from the programming model. It can be read or written through the BDM port using the RDMREG and WDMREG commands.

### 31.4.1 Revision A Shared Debug Resources

In the Revision A implementation of the debug module, certain hardware structures are shared between BDM and breakpoint functionality as shown in [Table 31-5](#).

**Table 31-5. Rev. A Shared BDM/Breakpoint Hardware**

Register	BDM Function	Breakpoint Function
AATR	Bus attributes for all memory commands	Attributes for address breakpoint
ABHR	Address for all memory commands	Address for address breakpoint
DBR	Data for all BDM write commands	Data for data breakpoint

Thus, loading a register to perform a specific function that shares hardware resources is destructive to the shared function. For example, a BDM command to access memory overwrites an address breakpoint in ABHR. A BDM write command overwrites the data breakpoint in DBR.

### 31.4.2 Address Attribute Trigger Register (AATR)

The AATR, shown in [Figure 31-5](#), defines address attributes and a mask to be matched in the trigger. The register value is compared with address attribute signals from the processor's local high-speed bus, as defined by the setting of the trigger definition register (TDR). AATR is accessible in supervisor mode as debug control register 0x06 using the WDEBUG instruction and through the BDM port using the WDMREG command.

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R																
W	RM	SZM		TTM		TMM		R	SZ		TT		TM			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1
DRc[4:0]	0x06															

**Figure 31-5. Address Attribute Trigger Register (AATR)**

**Table 31-6. AATR Field Descriptions**

Bits	Name	Description																																				
15	RM	Read/write mask. Setting RM masks R in address comparisons.																																				
14–13	SZM	Size mask. Setting an SZM bit masks the corresponding SZ bit in address comparisons.																																				
12–11	TTM	Transfer type mask. Setting a TTM bit masks the corresponding TT bit in address comparisons.																																				
10–8	TMM	Transfer modifier mask. Setting a TMM bit masks the corresponding TM bit in address comparisons.																																				
7	R	Read/write. R is compared with the R/W signal of the processor's local bus.																																				
6–5	SZ	Size. Compared to the processor's local bus size signals. 00 Longword 01 Byte 10 Word 11 Reserved																																				
4–3	TT	Transfer type. Compared with the local bus transfer type signals. 00 Normal processor access 01 Reserved 10 Emulator mode access 11 Acknowledge/CPU space access These bits also define the TT encoding for BDM memory commands. In this case, the 01 encoding indicates an external or DMA access (for backward compatibility). These bits affect the TM bits.																																				
2–0	TM	Transfer modifier. Compared with the local bus transfer modifier signals, which give supplemental information for each transfer type. These bits also define the TM encoding for BDM memory commands (for backward compatibility). <table><tr><th>TM</th><th>TT=00 (normal mode)</th><th>TT=10 (emulator mode)</th><th>TT=11 (acknowledge/CPU space transfers)</th></tr><tr><td>000</td><td>Explicit cach line push</td><td>Reserved</td><td>CPU space access</td></tr><tr><td>001</td><td>User data access</td><td>Reserved</td><td>Interrupt ack level 1</td></tr><tr><td>010</td><td>User code access</td><td>Reserved</td><td>Interrupt ack level 2</td></tr><tr><td>011</td><td>Reserved</td><td>Reserved</td><td>Interrupt ack level 3</td></tr><tr><td>100</td><td>Reserved</td><td>Reserved</td><td>Interrupt ack level 4</td></tr><tr><td>101</td><td>Supervisor data access</td><td>Emulator mode access</td><td>Interrupt ack level 5</td></tr><tr><td>110</td><td>Supervisor code access</td><td>Emulator code Access</td><td>Interrupt ack level 6</td></tr><tr><td>111</td><td>Reserved</td><td>Reserved</td><td>Interrupt ack level 7</td></tr></table>	TM	TT=00 (normal mode)	TT=10 (emulator mode)	TT=11 (acknowledge/CPU space transfers)	000	Explicit cach line push	Reserved	CPU space access	001	User data access	Reserved	Interrupt ack level 1	010	User code access	Reserved	Interrupt ack level 2	011	Reserved	Reserved	Interrupt ack level 3	100	Reserved	Reserved	Interrupt ack level 4	101	Supervisor data access	Emulator mode access	Interrupt ack level 5	110	Supervisor code access	Emulator code Access	Interrupt ack level 6	111	Reserved	Reserved	Interrupt ack level 7
TM	TT=00 (normal mode)	TT=10 (emulator mode)	TT=11 (acknowledge/CPU space transfers)																																			
000	Explicit cach line push	Reserved	CPU space access																																			
001	User data access	Reserved	Interrupt ack level 1																																			
010	User code access	Reserved	Interrupt ack level 2																																			
011	Reserved	Reserved	Interrupt ack level 3																																			
100	Reserved	Reserved	Interrupt ack level 4																																			
101	Supervisor data access	Emulator mode access	Interrupt ack level 5																																			
110	Supervisor code access	Emulator code Access	Interrupt ack level 6																																			
111	Reserved	Reserved	Interrupt ack level 7																																			

### 31.4.3 Address Breakpoint Registers (ABLR, ABHR)

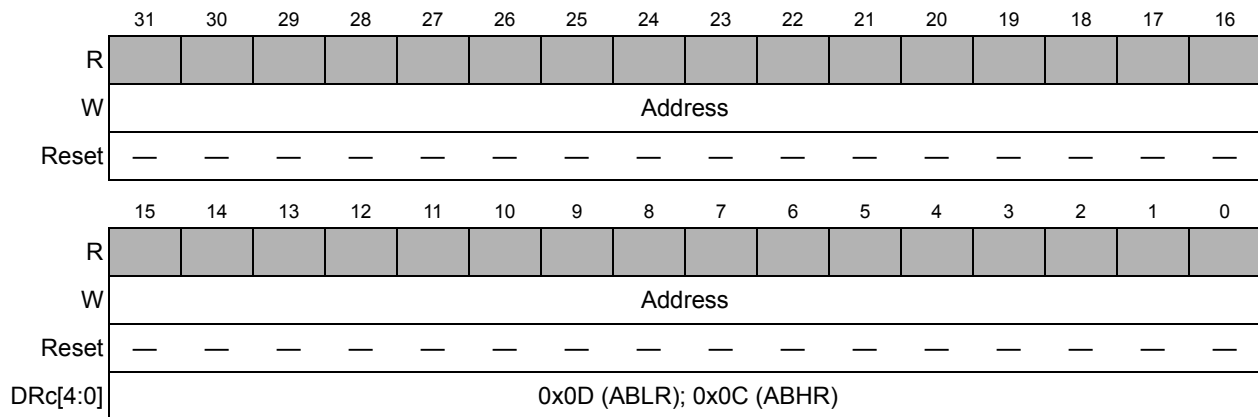
The ABLR and ABHR, shown in [Figure 31-6](#), define regions in the processor's data address space that can be used as part of the trigger. These register values are compared with the address for each



transfer on the processor's high-speed local bus. The trigger definition register (TDR) identifies the trigger as one of three cases:

1. Identical to the value in ABLR
2. Inside the range bound by ABLR and ABHR inclusive
3. Outside that same range

ABHR is accessible in supervisor mode as debug control register 0x0C using the WDEBUG instruction and via the BDM port using the RDMREG and WDMREG commands. ABLR is accessible in supervisor mode as debug control register 0x0D using the WDEBUG instruction and via the BDM port using the WDMREG command.



**Figure 31-6. Address Breakpoint Registers (ABLR, ABHR)**

**Table 31-7. ABLR Field Description**

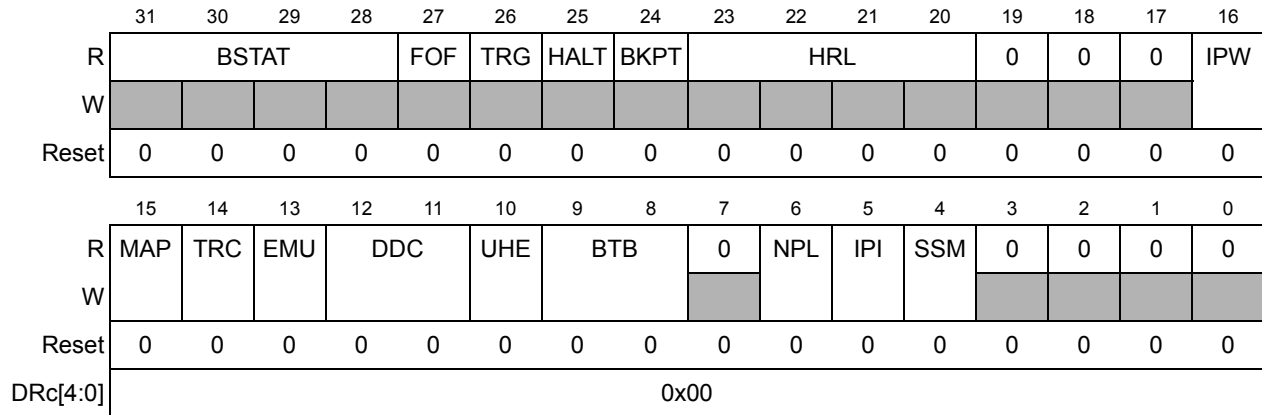
Bits	Name	Description
31–0	Address	Low address. Holds the 32-bit address marking the lower bound of the address breakpoint range. Breakpoints for specific addresses are programmed into ABLR.

**Table 31-8. ABHR Field Description**

Bits	Name	Description
31–0	Address	High address. Holds the 32-bit address marking the upper bound of the address breakpoint range.

### 31.4.4 Configuration/Status Register (CSR)

The CSR defines the debug configuration for the processor and memory subsystem and contains status information from the breakpoint logic. CSR is write-only from the programming model. It can be read from and written to through the BDM port. CSR is accessible in supervisor mode as debug control register 0x00 using the WDEBUG instruction and through the BDM port using the RDMREG and WDMREG commands.

**Figure 31-7. Configuration/Status Register (CSR)****Table 31-9. CSR Field Descriptions**

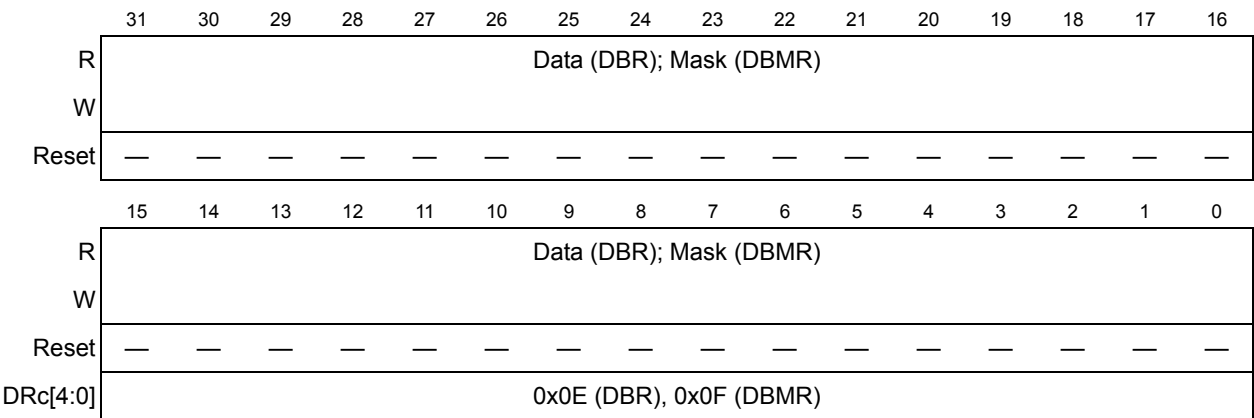
Bits	Name	Description
31–28	BSTAT	Breakpoint status. Provides read-only status information concerning hardware breakpoints. BSTAT is cleared by a TDR write or by a CSR read when either a level-2 breakpoint is triggered or a level-1 breakpoint is triggered and the level-2 breakpoint is disabled. 0000 No breakpoints enabled 0001 Waiting for level-1 breakpoint 0010 Level-1 breakpoint triggered 0101 Waiting for level-2 breakpoint 0110 Level-2 breakpoint triggered
27	FOF	Fault-on-fault. If FOF is set, a catastrophic halt occurred and forced entry into BDM. FOF is cleared whenever CSR is read.
26	TRG	Hardware breakpoint trigger. If TRG is set, a hardware breakpoint halted the processor core and forced entry into BDM. Reset, the debug GO command, or reading CSR will clear TRG.
25	HALT	Processor halt. If HALT is set, the processor executed a HALT and forced entry into BDM. Reset, the debug GO command, or reading CSR will clear HALT.
24	BKPT	Breakpoint assert. If BKPT is set, BKPT was asserted, forcing the processor into BDM. Reset, the debug GO command, or reading CSR will clear BKPT.
23–20	HRL	Hardware revision level. Indicates the level of debug module functionality. An emulator could use this information to identify the level of functionality supported. 0000 Initial debug functionality (Revision A) (This is the only valid value for the MCF5275)
19–17	—	Reserved, should be cleared.
16	IPW	Inhibit processor writes. Setting IPW inhibits processor-initiated writes to the debug module's programming model registers. IPW can be modified only by commands from the external development system.
15	MAP	Force processor references in emulator mode. 0 All emulator-mode references are mapped into supervisor code and data spaces. 1 The processor maps all references while in emulator mode to a special address space, TT = 10, TM = 101 or 110.
14	TRC	Force emulation mode on trace exception. If TRC = 1, the processor enters emulator mode when a trace exception occurs. If TRC=0, the processor enters supervisor mode.

**Table 31-9. CSR Field Descriptions (Continued)**

Bits	Name	Description
13	EMU	Force emulation mode. If EMU = 1, the processor begins executing in emulator mode. See <a href="#">Section 31.6.1.1, “Emulator Mode.”</a>
12–11	DDC	Debug data control. Controls operand data capture for DDATA, which displays the number of bytes defined by the operand reference size before the actual data; byte displays 8 bits, word displays 16 bits, and long displays 32 bits (one nibble at a time across multiple PSTCLK cycles). See <a href="#">Table 31-2</a> . 00 No operand data is displayed. 01 Capture all write data. 10 Capture all read data. 11 Capture all read and write data.
10	UHE	User halt enable. Selects the CPU privilege level required to execute the HALT instruction. 0 HALT is a supervisor-only instruction. 1 HALT is a supervisor/user instruction.
9–8	BTB	Branch target bytes. Defines the number of bytes of branch target address DDATA displays. 00 0 bytes 01 Lower 2 bytes of the target address 10 Lower 3 bytes of the target address 11 Entire 4-byte target address See <a href="#">Section 31.3.1, “Begin Execution of Taken Branch (PST = 0x5).”</a>
7	—	Reserved, should be cleared.
6	NPL	Non-pipelined mode. Determines whether the core operates in pipelined or mode or not. 0 Pipelined mode 1 Nonpipelined mode. The processor effectively executes one instruction at a time with no overlap. This adds at least 5 cycles to the execution time of each instruction. Given an average execution latency of 1.6 cycles/instruction, throughput in non-pipeline mode would be 6.6 cycles/instruction, approximately 25% or less of pipelined performance. Regardless of the NPL state, a triggered PC breakpoint is always reported before the triggering instruction executes. In normal pipeline operation, the occurrence of an address and/or data breakpoint trigger is imprecise. In non-pipeline mode, triggers are always reported before the next instruction begins execution and trigger reporting can be considered precise.
5	IPI	Ignore pending interrupts. 1 Core ignores any pending interrupt requests signalled while in single-instruction-step mode. 0 Core services any pending interrupt requests that were signalled while in single-step mode.
4	SSM	Single-step mode. Setting SSM puts the processor in single-step mode. 0 Normal mode. 1 Single-step mode. The processor halts after execution of each instruction. While halted, any BDM command can be executed. On receipt of the GO command, the processor executes the next instruction and halts again. This process continues until SSM is cleared.
3–0	—	Reserved, should be cleared.

# 31.4.5 Data Breakpoint/Mask Registers (DBR, DBMR)

The DBR, shown in [Figure 31-8](#), specifies data patterns used as part of the trigger into debug mode. DBR bits are masked by setting corresponding DBMR bits, as defined in TDR. DBR is accessible in supervisor mode as debug control register 0x0E, using the WDEBUG instruction and through the BDM port using the RDMREG and WDMREG commands. DBMR is accessible in supervisor mode as debug control register 0x0F, using the WDEBUG instruction and via the BDM port using the WDMREG command.



**Figure 31-8. Data Breakpoint/Mask Registers (DBR/DBMR)**

**Table 31-10. DBR Field Descriptions**

Bits	Name	Description
31–0	Data	Data breakpoint value. Contains the value to be compared with the data value from the processor's local bus as a breakpoint trigger.

**Table 31-11. DBMR Field Descriptions**

Bits	Name	Description
31–0	Mask	Data breakpoint mask. The 32-bit mask for the data breakpoint trigger. Clearing a DBR bit allows the corresponding DBR bit to be compared to the appropriate bit of the processor's local data bus. Setting a DBMR bit causes that bit to be ignored.

The DBR supports both aligned and misaligned references. [Table 31-12](#) shows relationships between processor address, access size, and location within the 32-bit data bus.

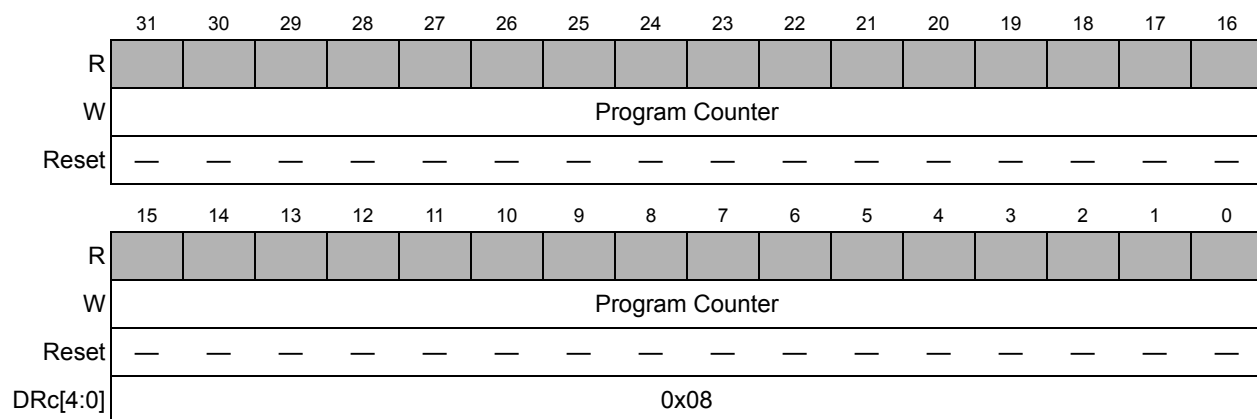
**Table 31-12. Access Size and Operand Data Location**

A[1:0]	Access Size	Operand Location
00	Byte	D[31:24]
01	Byte	D[23:16]
10	Byte	D[15:8]
11	Byte	D[7:0]
0x	Word	D[31:16]
1x	Word	D[15:0]
xx	Longword	D[31:0]

### 31.4.6 Program Counter Breakpoint/Mask Registers (PBR, PBMR)

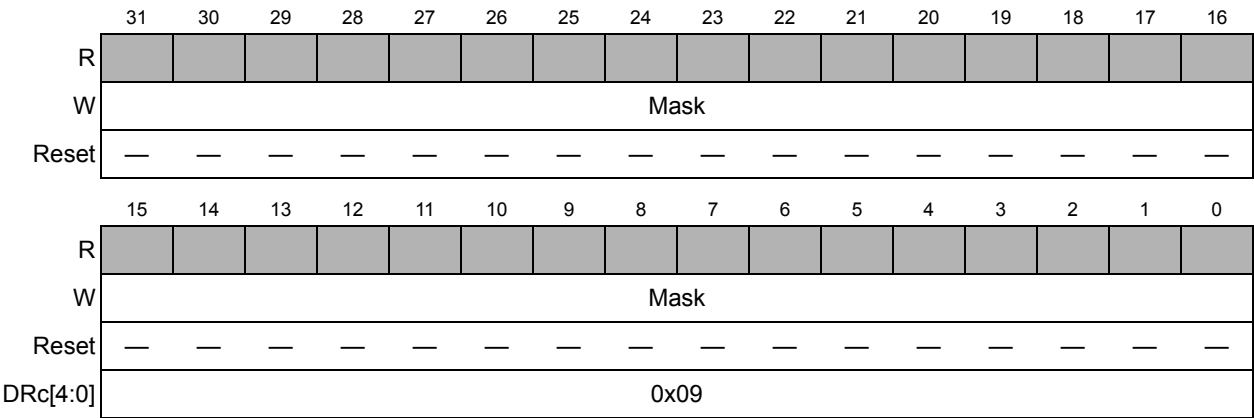
The PBR defines an instruction address for use as part of the trigger. This register's contents are compared with the processor's program counter register when TDR is configured appropriately. PBR bits are masked by setting corresponding PBMR bits. Results are compared with the processor's program counter register, as defined in TDR. [Figure 31-9](#) shows the PC breakpoint register.

The PC breakpoint register is accessible in supervisor mode using the WDEBBUG instruction and through the BDM port using the RDMREG and WDMREG commands using values shown in [Section 31.5.3.3, "Command Set Descriptions."](#)

**Figure 31-9. Program Counter Breakpoint Register (PBR)****Table 31-13. PBR Field Descriptions**

Bits	Name	Description
31–0	Address	PC breakpoint address. The 32-bit address to be compared with the PC as a breakpoint trigger.

Figure 31-9 shows PBMR. PBMR is accessible in supervisor mode as debug control register 0x09 using the WDEBUG instruction and via the BDM port using the wdmreg command.



**Figure 31-10. Program Counter Breakpoint Mask Register (PBMR)**

**Table 31-14. PBMR Field Descriptions**

Bits	Name	Description
31–0	Mask	PC breakpoint mask. A zero in a bit position causes the corresponding PBR bit to be compared to the appropriate PC bit. Set PBMR bits cause PBR bits to be ignored.

### 31.4.7 Trigger Definition Register (TDR)

The TDR, shown in Table 31-11, configures the operation of the hardware breakpoint logic that corresponds with the ABHR/ABLR/AATR, PBR/PBMR, and DBR/DBMR registers within the debug module. The TDR controls the actions taken under the defined conditions. Breakpoint logic may be configured as a one- or two-level trigger. TDR[31–16] define the second-level trigger and bits 15–0 define the first-level trigger.

#### NOTE

The debug module has no hardware interlocks, so to prevent spurious breakpoint triggers while the breakpoint registers are being loaded, disable TDR (by clearing TDR[29,13]) before defining triggers.

A write to TDR clears the CSR trigger status bits, CSR[BSTAT]. TDR is accessible in supervisor mode as debug control register 0x07 using the WDEBUG instruction and through the BDM port using the WDMREG command.

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R																
W	TRC		EBL	EDLW	EDWL	EDWU	EDLL	EDLM	EDUM	EDUU	DI	EAI	EAR	EAL	EPC	PCI
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																
W	0	0	EBL	EDLW	EDWL	EDWU	EDLL	EDLM	EDUM	EDUU	DI	EAI	EAR	EAL	EPC	PCI
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

DRc[4:0] 0x07

Figure 31-11. Trigger Definition Register (TDR)

Table 31-15. TDR Field Descriptions

Bits	Name	Description
31–30	TRC	Trigger response control. Determines how the processor responds to a completed trigger condition. The trigger response is always displayed on DDATA. 00 Display on DDATA only 01 Processor halt 10 Debug interrupt 11 Reserved
15–14	—	Reserved, should be cleared.
29/13	EBL	Enable breakpoint. Global enable for the breakpoint trigger. Setting TDR[EBL] enables a breakpoint trigger. Clearing it disables all breakpoints at that level.
28–22/ 12–6	EDx	Setting an EDx bit enables the corresponding data breakpoint condition based on the size and placement on the processor's local data bus. Clearing all EDx bits disables data breakpoints.
28/12		EDLW Data longword. Entire processor's local data bus.
27/11		EDWL Lower data word.
26/10		EDWU Upper data word.
25/9		EDLL Lower lower data byte. Low-order byte of the low-order word.
24/8		EDLM Lower middle data byte. High-order byte of the low-order word.
23/7		EDUM Upper middle data byte. Low-order byte of the high-order word.
22/6		EDUU Upper upper data byte. High-order byte of the high-order word.
21/5	DI	Data breakpoint invert. Provides a way to invert the logical sense of all the data breakpoint comparators. This can develop a trigger based on the occurrence of a data value other than the DBR contents.

**Table 31-15. TDR Field Descriptions (Continued)**

Bits	Name	Description	
20–18/4–2	EAX	Enable address bits. Setting an EA bit enables the corresponding address breakpoint. Clearing all three bits disables the breakpoint.	
20/4		EAI	Enable address breakpoint inverted. Breakpoint is based outside the range between ABLR and ABHR.
19/3		EAR	Enable address breakpoint range. The breakpoint is based on the inclusive range defined by ABLR and ABHR.
18/2		EAL	Enable address breakpoint low. The breakpoint is based on the address in the ABLR.
17/1	EPC	Enable PC breakpoint. If set, this bit enables the PC breakpoint.	
16/0	PCI	Breakpoint invert. If set, this bit allows execution outside a given region as defined by PBR and PBMR to enable a trigger. If cleared, the PC breakpoint is defined within the region defined by PBR and PBMR.	

## 31.5 Background Debug Mode (BDM)

The ColdFire Family implements a low-level system debugger in the microprocessor hardware. Communication with the development system is handled through a dedicated, high-speed serial command interface. The ColdFire architecture implements the BDM controller in a dedicated hardware module. Although some BDM operations, such as CPU register accesses, require the CPU to be halted, other BDM commands, such as memory accesses, can be executed while the processor is running.

### 31.5.1 CPU Halt

Although most BDM operations can occur in parallel with CPU operations, unrestricted BDM operation requires the CPU to be halted. The sources that can cause the CPU to halt are listed below in order of priority:

1. A catastrophic fault-on-fault condition automatically halts the processor.
2. A hardware breakpoint can be configured to generate a pending halt condition similar to the assertion of  $\overline{\text{BKPT}}$ . This type of halt is always first made pending in the processor. Next, the processor samples for pending halt and interrupt conditions once per instruction. When a pending condition is asserted, the processor halts execution at the next sample point. See [Section 31.6.1, “Theory of Operation.”](#)
3. The execution of a HALT instruction immediately suspends execution. Attempting to execute HALT in user mode while  $\text{CSR}[\text{UHE}] = 0$  generates a privilege violation exception. If  $\text{CSR}[\text{UHE}] = 1$ , HALT can be executed in user mode. After HALT executes, the processor can be restarted by serial shifting a GO command into the debug module. Execution continues at the instruction after HALT.



4. The assertion of the  $\overline{\text{BKPT}}$  input is treated as a pseudo-interrupt; that is, the halt condition is postponed until the processor core samples for halts/interrupts. The processor samples for these conditions once during the execution of each instruction. If there is a pending halt condition at the sample time, the processor suspends execution and enters the halted state.

The assertion of  $\overline{\text{BKPT}}$  should be considered in the following two special cases:

- After the system reset signal is negated, the processor waits for 16 processor clock cycles before beginning reset exception processing. If the  $\overline{\text{BKPT}}$  input is asserted within eight cycles after  $\overline{\text{RESET}}$  is negated, the processor enters the halt state, signaling halt status (0xF) on the PST outputs. While the processor is in this state, all resources accessible through the debug module can be referenced. This is the only chance to force the processor into emulation mode through CSR[EMU].

After system initialization, the processor's response to the GO command depends on the set of BDM commands performed while it is halted for a breakpoint. Specifically, if the PC register was loaded, the GO command causes the processor to exit halted state and pass control to the instruction address in the PC, bypassing normal reset exception processing. If the PC was not loaded, the GO command causes the processor to exit halted state and continue reset exception processing.

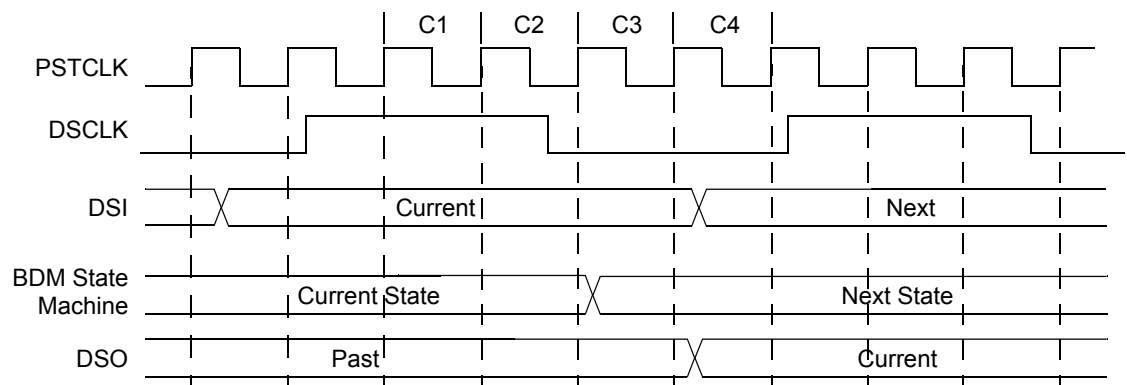
- The ColdFire architecture also handles a special case of  $\overline{\text{BKPT}}$  being asserted while the processor is stopped by execution of the STOP instruction. For this case, the processor exits the stopped mode and enters the halted state, at which point, all BDM commands may be exercised. When restarted, the processor continues by executing the next sequential instruction, that is, the instruction following the STOP opcode.

The CSR[27–24] bits indicate the halt source, showing the highest priority source for multiple halt conditions.

### 31.5.2 BDM Serial Interface

When the CPU is halted and PST reflects the halt status, the development system can send unrestricted commands to the debug module. The debug module implements a synchronous protocol using two inputs (DSCLK and DSI) and one output (DSO), where DSO is specified as a delay relative to the rising edge of the processor clock. See [Table 31-1](#). The development system serves as the serial communication channel master and must generate DSCLK.

The serial channel operates at a frequency from DC to 1/5 of the PSTCLK frequency. The channel uses full-duplex mode, where data is sent and received simultaneously by both master and slave devices. The transmission consists of 17-bit packets composed of a status/control bit and a 16-bit data word. As shown in [Figure 31-12](#), all state transitions are enabled on a rising edge of PSTCLK when DSCLK is high; that is, DSI is sampled and DSO is driven.



**Figure 31-12. BDM Serial Interface Timing**

DSCLK and DSI are synchronized inputs. DSCLK acts as a pseudo clock enable and is sampled on the rising edge of the processor clock as well as the DSI. DSO is delayed from the DSCLK-enabled CLK rising edge (registered after a BDM state machine state change). All events in the debug module's serial state machine are based on the processor clock rising edge. DSCLK must also be sampled low (on a positive edge of CLK) between each bit exchange. The msb is transferred first. Because DSO changes state based on an internally-recognized rising edge of DSCLK, DSO cannot be used to indicate the start of a serial transfer. The development system must count clock cycles in a given transfer. C1–C4 are described as follows:

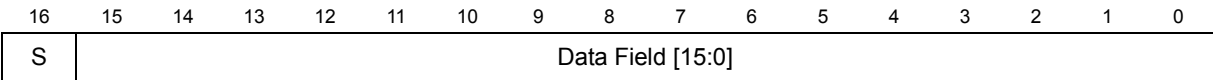
- C1—First synchronization cycle for DSI (DSCLK is high).
- C2—Second synchronization cycle for DSI (DSCLK is high).
- C3—BDM state machine changes state depending upon DSI and whether the entire input data transfer has been transmitted.
- C4—DSO changes to next value.

**NOTE**

A not-ready response can be ignored except during a memory-referencing cycle. Otherwise, the debug module can accept a new serial transfer after 32 processor clock periods.

**31.5.2.1 Receive Packet Format**

The basic receive packet, [Figure 31-13](#), consists of 16 data bits and 1 status bit



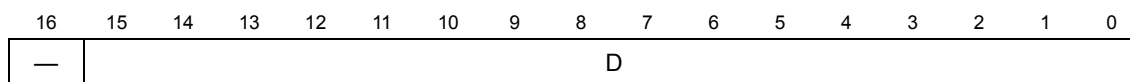
**Figure 31-13. Receive BDM Packet**

**Table 31-16. Receive BDM Packet Field Description**

Bits	Name	Description																		
16	S	<p>Status. Indicates the status of CPU-generated messages listed below. The not-ready response can be ignored unless a memory-referencing cycle is in progress. Otherwise, the debug module can accept a new serial transfer after 32 processor clock periods.</p> <table border="1"> <thead> <tr> <th>S</th><th>Data</th><th>Message</th></tr> </thead> <tbody> <tr> <td>0</td><td>xxxx</td><td>Valid data transfer</td></tr> <tr> <td>0</td><td>FFFF</td><td>Status OK</td></tr> <tr> <td>1</td><td>0000</td><td>Not ready with response; come again</td></tr> <tr> <td>1</td><td>0001</td><td>Error—Terminated bus cycle; data invalid</td></tr> <tr> <td>1</td><td>FFFF</td><td>Illegal Command</td></tr> </tbody> </table>	S	Data	Message	0	xxxx	Valid data transfer	0	FFFF	Status OK	1	0000	Not ready with response; come again	1	0001	Error—Terminated bus cycle; data invalid	1	FFFF	Illegal Command
S	Data	Message																		
0	xxxx	Valid data transfer																		
0	FFFF	Status OK																		
1	0000	Not ready with response; come again																		
1	0001	Error—Terminated bus cycle; data invalid																		
1	FFFF	Illegal Command																		
15–0	D	Data. Contains the message to be sent from the debug module to the development system. The response message is always a single word, with the data field encoded as shown above.																		

### 31.5.2.2 Transmit Packet Format

The basic transmit packet, [Figure 31-14](#), consists of 16 data bits and 1 reserved bit.

**Figure 31-14. Transmit BDM Packet****Table 31-17. Transmit BDM Packet Field Description**

Bits	Name	Description
16	—	Reserved, should be cleared.
15–0	D	Data bits 15–0. Contains the data to be sent from the development system to the debug module.

### 31.5.3 BDM Command Set

[Table 31-18](#) summarizes the BDM command set. Subsequent paragraphs contain detailed descriptions of each command. Issuing a BDM command when the processor is accessing debug module registers using the WDEBUD instruction causes undefined behavior.

**Table 31-18. BDM Command Summary**

Command	Mnemonic	Description	CPU State <sup>1</sup>	Section	Command (Hex)
Read A/D register	RAREG/ RDREG	Read the selected address or data register and return the results through the serial interface.	Halted	31.5.3.3.1	0x218 {A/D, Reg[2:0]}
Write A/D register	WAREG/ WDREG	Write the data operand to the specified address or data register.	Halted	31.5.3.3.2	0x208 {A/D, Reg[2:0]}
Read memory location	READ	Read the data at the memory location specified by the longword address.	Steal	31.5.3.3.3	0x1900—byte 0x1940—word 0x1980—lword <sup>2</sup>
Write memory location	WRITE	Write the operand data to the memory location specified by the longword address.	Steal	31.5.3.3.4	0x1800—byte 0x1840—word 0x1880—lword <sup>2</sup>
Dump memory block	DUMP	Used with READ to dump large blocks of memory. An initial READ is executed to set up the starting address of the block and to retrieve the first result. A DUMP command retrieves subsequent operands.	Steal	31.5.3.3.5	0x1D00—byte 0x1D40—word 0x1D80—lword <sup>2</sup>
Fill memory block	FILL	Used with WRITE to fill large blocks of memory. An initial WRITE is executed to set up the starting address of the block and to supply the first operand. A FILL command writes subsequent operands.	Steal	31.5.3.3.6	0x1C00—byte 0x1C40—word 0x1C80—lword <sup>2</sup>
Resume execution	GO	The pipeline is flushed and refilled before resuming instruction execution at the current PC.	Halted	31.5.3.3.7	0x0C00
No operation	NOP	Perform no operation; may be used as a null command.	Parallel	31.5.3.3.8	0x0000
Read control register	RCREG	Read the system control register.	Halted	31.5.3.3.9	0x2980
Write control register	WCREG	Write the operand data to the system control register.	Halted	31.5.3.3.10	0x2880
Read debug module register	RDMREG	Read the debug module register.	Parallel	31.5.3.3.11	0x2D {0x4 <sup>3</sup> DRc[4:0]}
Write debug module register	WDMREG	Write the operand data to the debug module register.	Parallel	31.5.3.3.12	0x2C {0x4 <sup>3</sup> DRc[4:0]}

<sup>1</sup> General command effect and/or requirements on CPU operation:

- Halted. The CPU must be halted to perform this command.
- Steal. Command generates bus cycles that can be interleaved with bus accesses.
- Parallel. Command is executed in parallel with CPU activity.

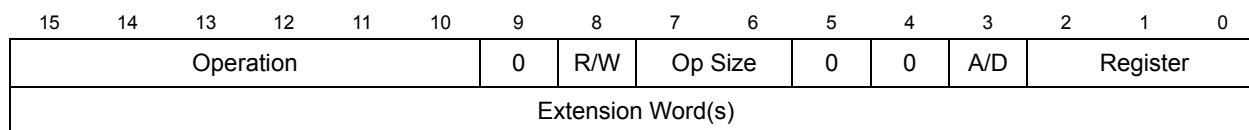
<sup>2</sup> lword = longword

<sup>3</sup> 0x4 is a three-bit field.

Unassigned command opcodes are reserved by Freescale. All unused command formats within any revision level perform a NOP and return the illegal command response.

### 31.5.3.1 ColdFire BDM Command Format

All ColdFire Family BDM commands include a 16-bit operation word followed by an optional set of one or more extension words, as shown in [Figure 31-15](#).



**Figure 31-15. BDM Command Format**

**Table 31-19. BDM Field Descriptions**

Bits	Name	Description															
15–10	Operation	Specifies the command. These values are listed in <a href="#">Table 31-18</a> .															
9	—	Reserved, should be cleared.															
8	R/W	Direction of operand transfer. 0 Data is written to the CPU or to memory from the development system. 1 The transfer is from the CPU to the development system.															
7–6	Op Size	Operand data size for sized operations. Addresses are expressed as 32-bit absolute values. Note that a command performing a byte-sized memory read leaves the upper 8 bits of the response data undefined. Referenced data is returned in the lower 8 bits of the response. <table border="1"> <tr> <th></th><th>Operand Size</th><th>Bit Values</th></tr> <tr> <td>00</td><td>Byte</td><td>8 bits</td></tr> <tr> <td>01</td><td>Word</td><td>16 bits</td></tr> <tr> <td>10</td><td>Longword</td><td>32 bits</td></tr> <tr> <td>11</td><td>Reserved</td><td>—</td></tr> </table>		Operand Size	Bit Values	00	Byte	8 bits	01	Word	16 bits	10	Longword	32 bits	11	Reserved	—
	Operand Size	Bit Values															
00	Byte	8 bits															
01	Word	16 bits															
10	Longword	32 bits															
11	Reserved	—															
5–4	—	Reserved, should be cleared.															
3	A/D	Address/data. Determines whether the register field specifies a data or address register. 0 Indicates a data register. 1 Indicates an address register.															
2–0	Register	Contains the register number in commands that operate on processor registers.															

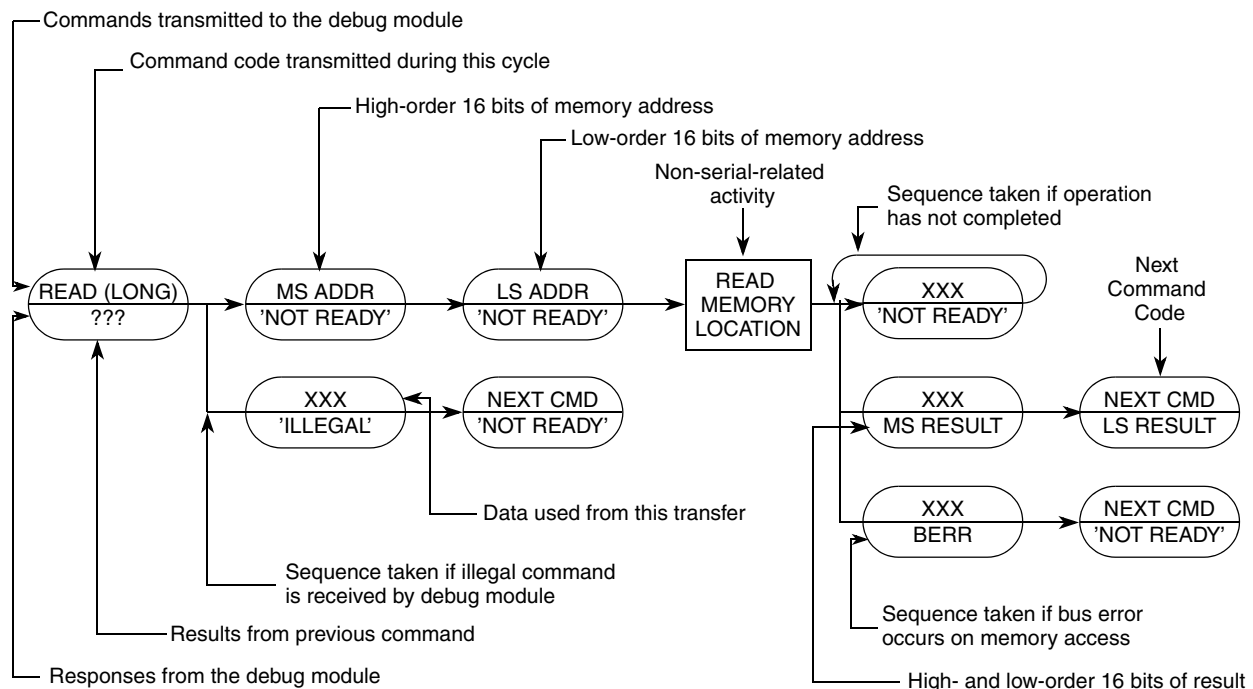
#### 31.5.3.1.1 Extension Words as Required

Some commands require extension words for addresses and/or immediate data. Addresses require two extension words because only absolute long addressing is permitted. Longword accesses are forcibly longword-aligned and word accesses are forcibly word-aligned. Immediate data can be 1 or 2 words long. Byte and word data each requires a single extension word and longword data requires two extension words.

Operands and addresses are transferred most-significant word first. In the following descriptions of the BDM command set, the optional set of extension words is defined as address, data, or operand data.

### 31.5.3.2 Command Sequence Diagrams

The command sequence diagram in Figure 31-16 shows serial bus traffic for commands. Each bubble represents a 17-bit bus transfer. The top half of each bubble indicates the data the development system sends to the debug module; the bottom half indicates the debug module's response to the previous development system commands. Command and result transactions overlap to minimize latency.



**Figure 31-16. Command Sequence Diagram**

The sequence is as follows:

- In cycle 1, the development system command is issued (READ in this example). The debug module responds with either the low-order results of the previous command or a command complete status of the previous command, if no results are required.
- In cycle 2, the development system supplies the high-order 16 address bits. The debug module returns a not-ready response unless the received command is decoded as unimplemented, which is indicated by the illegal command encoding. If this occurs, the development system should retransmit the command.

**NOTE**

A not-ready response can be ignored except during a memory-referencing cycle. Otherwise, the debug module can accept a new serial transfer after 32 processor clock periods.

- In cycle 3, the development system supplies the low-order 16 address bits. The debug module always returns a not-ready response.
- At the completion of cycle 3, the debug module initiates a memory read operation. Any serial transfers that begin during a memory access return a not-ready response.
- Results are returned in the two serial transfer cycles after the memory access completes. For any command performing a byte-sized memory read operation, the upper 8 bits of the response data are undefined and the referenced data is returned in the lower 8 bits. The next command's opcode is sent to the debug module during the final transfer. If a memory or register access is terminated with a bus error, the error status (S = 1, DATA = 0x0001) is returned instead of result data.

### 31.5.3.3 Command Set Descriptions

The following sections describe the commands summarized in [Table 31-18](#).

**NOTE**

The BDM status bit (S) is 0 for normally completed commands; S = 1 for illegal commands, not-ready responses, and transfers with bus-errors. [Section 31.5.2, “BDM Serial Interface,”](#) describes the receive packet format.

Freescale reserves unassigned command opcodes for future expansion. Unused command formats in any revision level perform a NOP and return an illegal command response.

#### 31.5.3.3.1 Read A/D Register (RAREG/RDREG)

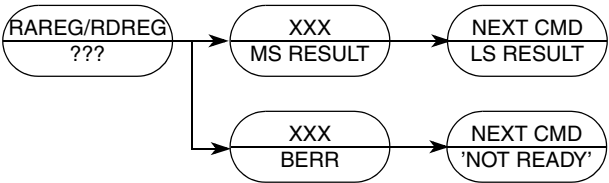
Read the selected address or data register and return the 32-bit result. A bus error response is returned if the CPU core is not halted.

Command/Result Formats:

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Command	0x2				0x1				0x8				A/D	Register		
Result	D[31:16]															
	D[15:0]															

**Figure 31-17. RAREG/RDREG Command Format**

Command Sequence:



**Figure 31-18. RAREG/RDREG Command Sequence**

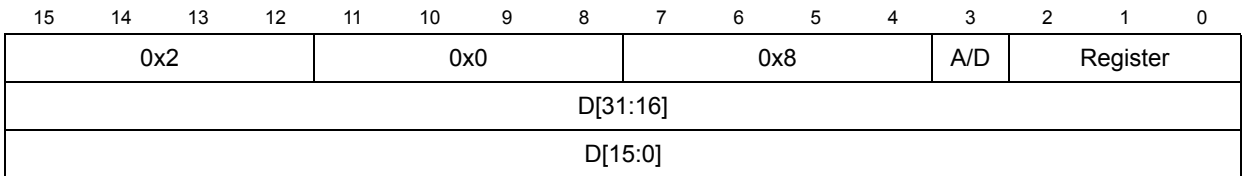
Operand Data:           None

Result Data:           The contents of the selected register are returned as a longword value, most-significant word first.

**31.5.3.3.2 Write A/D Register (WAREG/WDREG)**

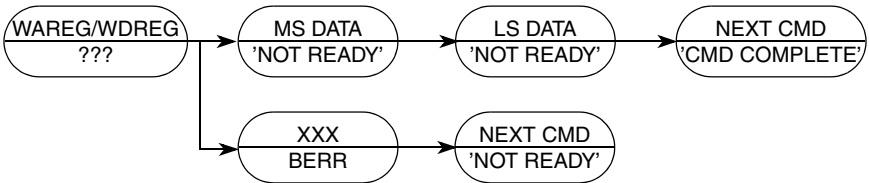
The operand longword data is written to the specified address or data register. A write alters all 32 register bits. A bus error response is returned if the CPU core is not halted.

Command Format:



**Figure 31-19. WAREG/WDREG Command Format**

Command Sequence



**Figure 31-20. WAREG/WDREG Command Sequence**

Operand Data           Longword data is written into the specified address or data register. The data is supplied most-significant word first.

Result Data            Command complete status is indicated by returning 0xFFFF (with S cleared) when the register write is complete.



### 31.5.3.3.3 Read Memory Location (READ)

Read data at the longword address. Address space is defined by BAAR[TT,TM]. Hardware forces low-order address bits to zeros for word and longword accesses to ensure that word addresses are word-aligned and longword addresses are longword-aligned.

Command/Result Formats:

		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Byte	Command	0x1				0x9				0x0				0x0			
		A[31:16]															
		A[15:0]															
	Result	X	X	X	X	X	X	X	X	D[7:0]							
Word	Command	0x1				0x9				0x4				0x0			
		A[31:16]															
		A[15:0]															
	Result	D[15:0]															
Longword	Command	0x1				0x9				0x8				0x0			
		A[31:16]															
		A[15:0]															
	Result	D[31:16]								D[15:0]							

Figure 31-21. READ Command/Result Formats

Command Sequence:

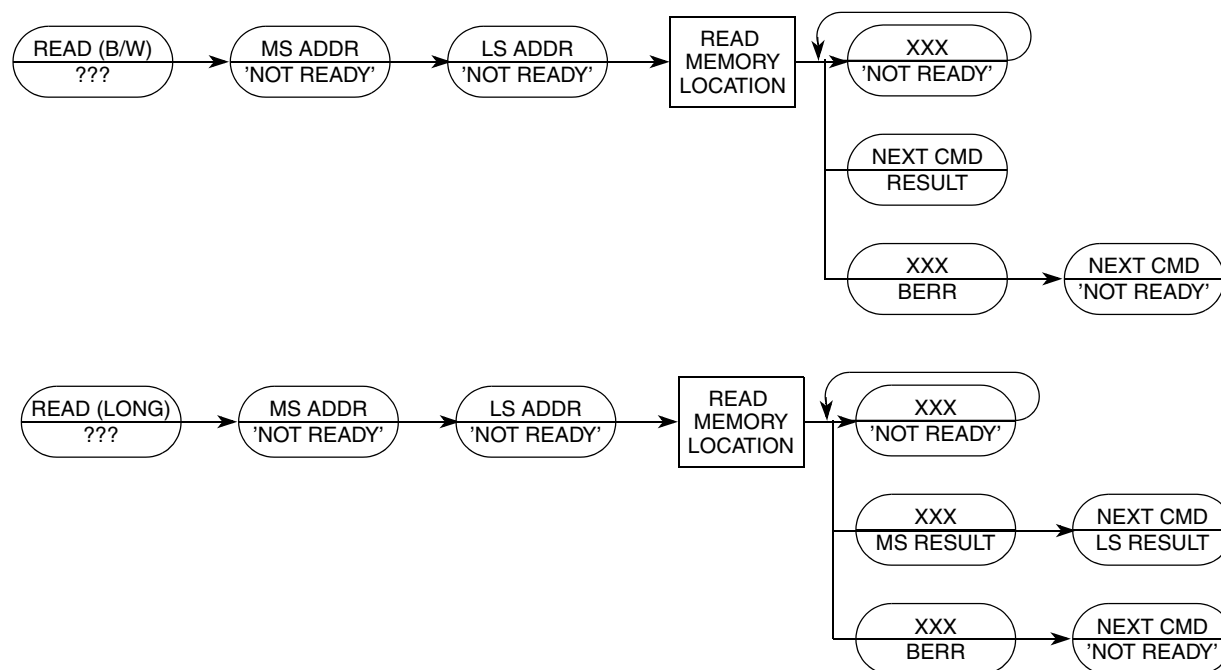


Figure 31-22. READ Command Sequence

**Operand Data**            The only operand is the longword address of the requested location.

**Result Data**            Word results return 16 bits of data; longword results return 32. Bytes are returned in the LSB of a word result; the upper byte is undefined. 0x0001 (S = 1) is returned if a bus error occurs.

**31.5.3.3.4 Write Memory Location (WRITE)**

Write data to the memory location specified by the longword address. The address space is defined by BAAR[TT,TM]. Hardware forces low-order address bits to zeros for word and longword accesses to ensure that word addresses are word-aligned and longword addresses are longword-aligned.

Command Formats:

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Byte	0x1				0x8				0x0				0x0			
	A[31:16]															
	A[15:0]															
	X	X	X	X	X	X	X	X	D[7:0]							
Word	0x1				0x8				0x4				0x0			
	A[31:16]															
	A[15:0]															
	D[15:0]															
Longword	0x1				0x8				0x8				0x0			
	A[31:16]															
	A[15:0]															
	D[31:16]															
	D[15:0]															

**Figure 31-23. WRITE Command Format**

## Command Sequence:

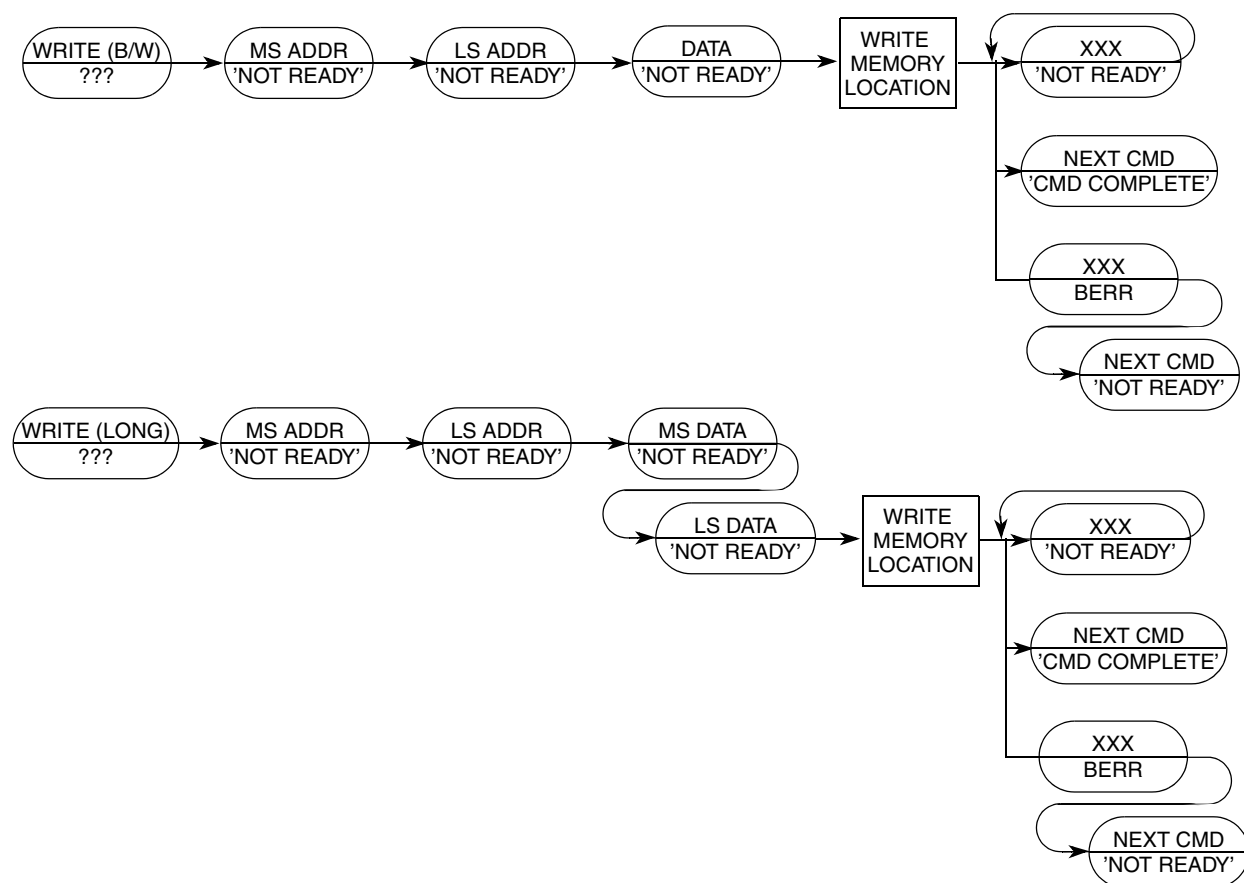


Figure 31-24. WRITE Command Sequence

Operand Data	This two-operand instruction requires a longword absolute address that specifies a location to which the data operand is to be written. Byte data is sent as a 16-bit word, justified in the LSB; 16- and 32-bit operands are sent as 16 and 32 bits, respectively
Result Data	Command complete status is indicated by returning 0xFFFF (with S cleared) when the register write is complete. A value of 0x0001 (with S set) is returned if a bus error occurs.

## 31.5.3.3.5 Dump Memory Block (DUMP)

DUMP is used with the READ command to access large blocks of memory. An initial READ is executed to set up the starting address of the block and to retrieve the first result. If an initial READ is not executed before the first DUMP, an illegal command response is returned. The DUMP command retrieves subsequent operands. The initial address is incremented by the operand size (1, 2, or 4) and saved in a temporary register. Subsequent DUMP commands use this address,

perform the memory read, increment it by the current operand size, and store the updated address in the temporary register.

**NOTE**

DUMP does not check for a valid address; it is a valid command only when preceded by NOP, READ, or another DUMP command. Otherwise, an illegal command response is returned. NOP can be used for intercommand padding without corrupting the address pointer.

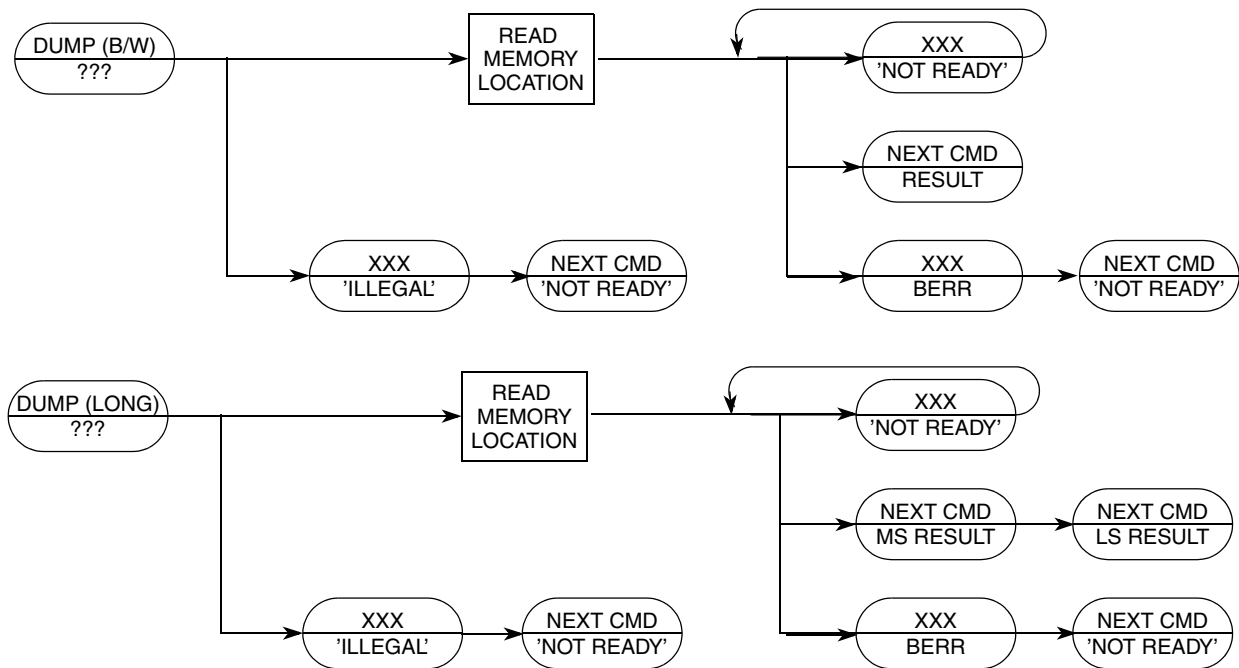
The size field is examined each time a DUMP command is processed, allowing the operand size to be dynamically altered.

Command/Result Formats:

		15		12		11		8		7		4		3		0
Byte	Command	0x1				0xD				0x0				0x0		
	Result	X	X	X	X	X	X	X	X	D[7:0]						
Word	Command	0x1				0xD				0x4				0x0		
	Result	D[15:0]														
Longword	Command	0x1				0xD				0x8				0x0		
	Result	D[31:16]														
		D[15:0]														

**Figure 31-25. DUMP Command/Result Formats**

Command Sequence:



**Figure 31-26. DUMP Command Sequence**

Operand Data: None

Result Data: Requested data is returned as either a word or longword. Byte data is returned in the least-significant byte of a word result. Word results return 16 bits of significant data; longword results return 32 bits. A value of 0x0001 (with S set) is returned if a bus error occurs.

### 31.5.3.3.6 Fill Memory Block (FILL)

A FILL command is used with the WRITE command to access large blocks of memory. An initial WRITE is executed to set up the starting address of the block and to supply the first operand. The FILL command writes subsequent operands. The initial address is incremented by the operand size (1, 2, or 4) and saved in a temporary register after the memory write. Subsequent FILL commands use this address, perform the write, increment it by the current operand size, and store the updated address in the temporary register.

If an initial WRITE is not executed preceding the first FILL command, the illegal command response is returned.

NOTE

The FILL command does not check for a valid address—FILL is a valid command only when preceded by another FILL, a NOP, or a WRITE command. Otherwise, an illegal command response is returned. The NOP command can be used for intercommand padding without corrupting the address pointer.

The size field is examined each time a FILL command is processed, allowing the operand size to be altered dynamically.

Command Formats:

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Byte	0x1				0xC				0x0				0x0			
	X	X	X	X	X	X	X	X	D[7:0]							
Word	0x1				0xC				0x4				0x0			
	D[15:0]															
Longword	0x1				0xC				0x8				0x0			
	D[31:16]															
	D[15:0]															

Figure 31-27. FILL Command Format

Command Sequence:

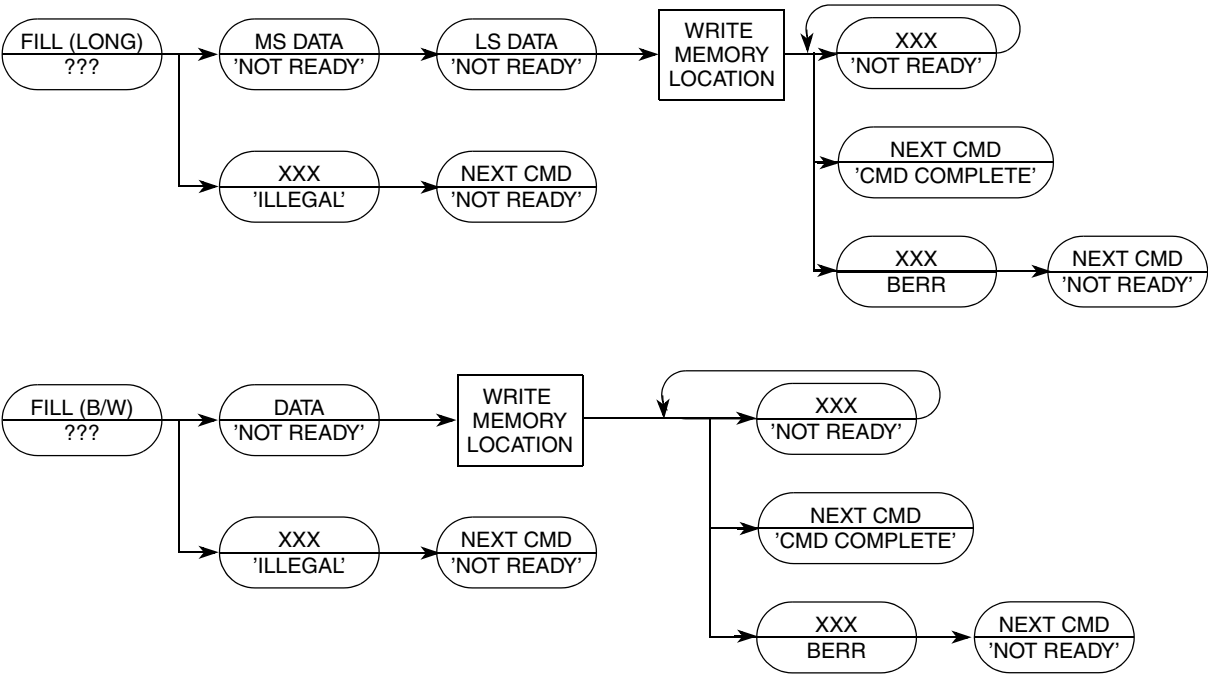


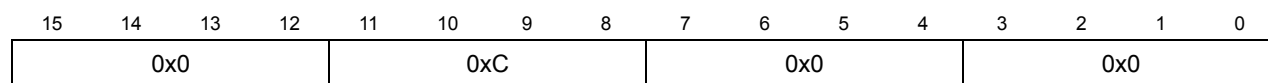
Figure 31-28. FILL Command Sequence

**Operand Data:** A single operand is data to be written to the memory location. Byte data is sent as a 16-bit word, justified in the least-significant byte; 16- and 32-bit operands are sent as 16 and 32 bits, respectively.

**Result Data:** Command complete status (0xFFFF) is returned when the register write is complete. A value of 0x0001 (with S set) is returned if a bus error occurs.

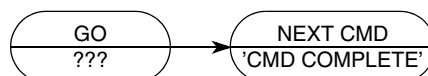
### 31.5.3.3.7 Resume Execution (GO)

The pipeline is flushed and refilled before normal instruction execution resumes. Prefetching begins at the current address in the PC and at the current privilege level. If any register (such as the PC or SR) is altered by a BDM command while the processor is halted, the updated value is used when prefetching resumes. If a GO command is issued and the CPU is not halted, the command is ignored.



**Figure 31-29. GO Command Format**

**Command Sequence:**



**Figure 31-30. GO Command Sequence**

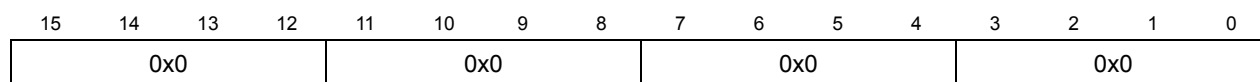
**Operand Data:** None

**Result Data:** The command-complete response (0xFFFF) is returned during the next shift operation.

### 31.5.3.3.8 No Operation (NOP)

NOP performs no operation and may be used as a null command where required.

**Command Formats:**



**Figure 31-31. NOP Command Format**

**Command Sequence:**

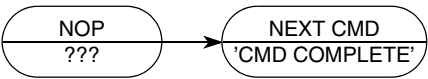


Figure 31-32. NOP Command Sequence

Operand Data:           None

Result Data:           The command-complete response, 0xFFFF (with S cleared), is returned during the next shift operation.

31.5.3.3.9 Read Control Register (RCREG)

Reads the selected control register and returns the 32-bit result. Accesses to the processor/memory control registers are always 32 bits wide, regardless of register width. The second and third words of the command form a 32-bit address, which the debug module uses to generate a special bus cycle to access the specified control register. The 12-bit Rc field is the same as that used by the MOVEC instruction.

Command/Result Formats:

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Command	0x2				0x9				0x8				0x0			
	0x0				0x0				0x0				0x0			
	0x0				Rc											
Result	D[31:16]															
	D[15:0]															

Figure 31-33. RCREG Command/Result Formats

Rc encoding:

Table 31-20. Control Register Map

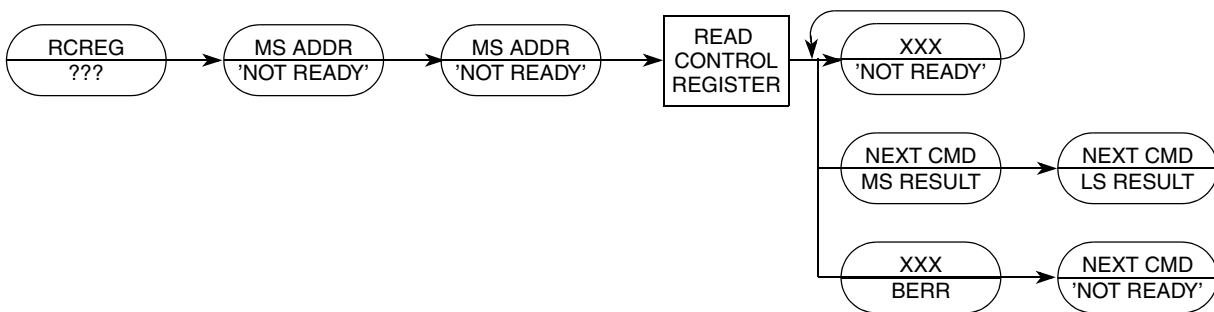
Rc	Register Definition
0x002	Cache Control Register (CACR)
0x004	Access Control Register (ACR0)
0x005	Access Control Register (ACR1)
0x800	Other Stack Pointer (OTHER_A7)
0x801	Vector Base Register (VBR)
0x804	MAC Status Register (MACSR)
0x805	MAC Mask Register (MASK)
0x806	MAC Accumulator 0 (ACC0)
0x807	MAC Accumulator 0,1 Extension Bytes (ACCEXT01)



**Table 31-20. Control Register Map**

Rc	Register Definition
0x808	MAC Accumulator 2,3 Extension Bytes (ACCEXT23)
0x809	MAC Accumulator 1 (ACC1)
0x80A	MAC Accumulator 2 (ACC2)
0x80B	MAC Accumulator 3 (ACC3)
0x80E	Status Register (SR)
0x80F	Program Register (PC)
0xC04	RAM Base Address Register 0 (RAMBAR0)

Command Sequence:

**Figure 31-34. RCREG Command Sequence**

**Operand Data:** The only operand is the 32-bit Rc control register select field.

**Result Data:** Control register contents are returned as a longword, most-significant word first. The implemented portion of registers smaller than 32 bits is guaranteed correct; other bits are undefined.

### BDM Accesses of the Stack Pointer Registers (A7: SSP, USP)

The V2 core of the MCF5275 supports two unique stack pointer (A7) registers: the supervisor stack pointer (SSP) and the user stack pointer (USP). The hardware implementation of these two programmable-visible 32-bit registers does not uniquely identify one as the SSP and the other as the USP. Rather, the hardware uses one 32-bit register as the currently-active A7 and the other register is named simply the “other\_A7”. Thus, the contents of the two hardware registers is a function of the operating mode of the processor:

```

if SR[S] = 1
    then      A7 = Supervisor Stack Pointer
              other_A7 = User Stack Pointer
    else      A7 = User Stack Pointer
              other_A7 = Supervisor Stack Pointer
  
```

The BDM programming model supports reads and writes to the A7 and other\_A7 registers directly. It is the responsibility of the external development system to determine the mapping of the two hardware registers (A7, other\_A7) to the two program-visible definitions (supervisor and user stack pointers), based on the Supervisor bit of the status register.

### BDM Accesses of the EMAC Registers

The presence of rounding logic in the output datapath of the EMAC requires that special care be taken during any BDM-initiated reads and writes of its programming model. In particular, it is necessary that any result rounding modes be disabled during the read/write process so the exact bit-wise contents of the EMAC registers are accessed.

As an example, any BDM read of an accumulator register (ACCn) must be preceded by two commands accessing the MAC status register. Specifically, the following BDM sequence is required:

```
BdmReadACCn (
    rcreg    macsr;           // read macsr contents & save
    wcreg    #0,macsr;       // disable all rounding modes
    rcreg    accn;           // read the desired accumulator
    wcreg    #saved_data,macsr; // restore the original macsr
)
```

Likewise, the following BDM sequence is needed to write an accumulator register:

```
BdmWriteACCn (
    rcreg    macsr;           // read macsr contents & save
    wcreg    #0,macsr;       // disable all rounding modes
    wcreg    #data,accn;     // write the desired accumulator
    wcreg    #saved_data,macsr; // restore the original macsr
)
```

Additionally, it is required that writes to the accumulator extension registers be performed after the corresponding accumulators are updated. This is needed since a write to any accumulator alters the contents of the corresponding extension register.

For more information on saving and restoring the complete EMAC programming model, see [Section 4.4.1.1.2, “Saving and Restoring the EMAC Programming Model.”](#)

### 31.5.3.3.10 Write Control Register (WCREG)

The operand (longword) data is written to the specified control register. The write alters all 32 register bits.

## Command/Result Formats:

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Command	0x2				0x8				0x8				0x0			
	0x0				0x0				0x0				0x0			
	0x0				Rc											
Result	D[31:16]															
	D[15:0]															

Figure 31-35. WCREG Command/Result Formats

## Command Sequence:

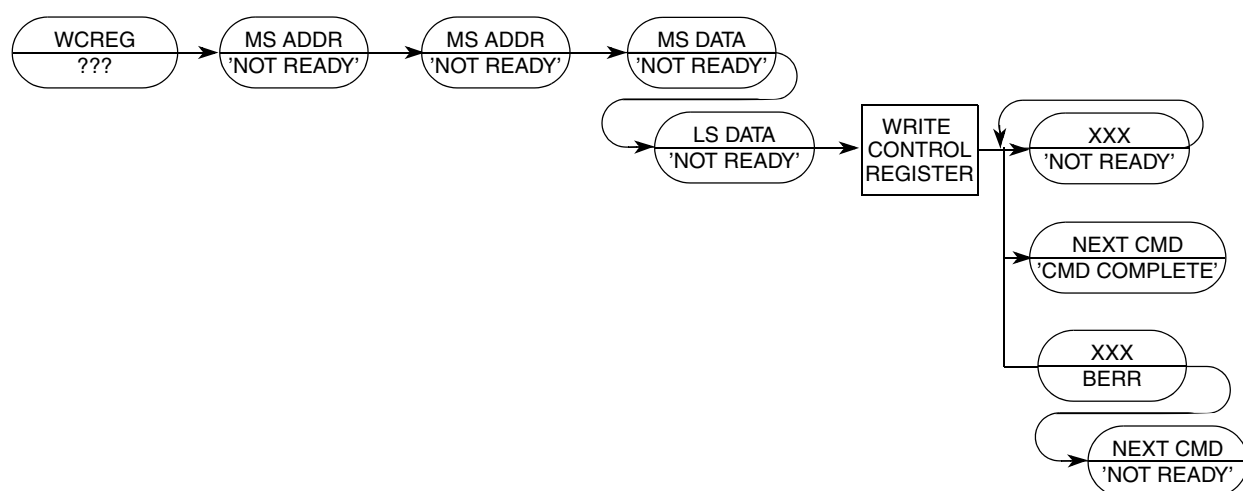


Figure 31-36. WCREG Command Sequence

**Operand Data:** This instruction requires two longword operands. The first selects the register to which the operand data is to be written; the second contains the data.

**Result Data:** Successful write operations return 0xFFFF. Bus errors on the write cycle are indicated by the setting of bit 16 in the status message and by a data pattern of 0x0001.

### 31.5.3.3.11 Read Debug Module Register (RDMREG)

Read the selected debug module register and return the 32-bit result. The only valid register selection for the RDMREG command is CSR (DRc = 0x00). Note that this read of the CSR clears CSR[FOF, TRG, HALT, BKPT]; as well as the trigger status bits (CSR[BSTAT]) if either a level-2 breakpoint has been triggered or a level-1 breakpoint has been triggered and no level-2 breakpoint has been enabled.

## Command/Result Formats:

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Command	0x2				0xD				100				DRc			
Result	D[31:16]															
	D[15:0]															

Figure 31-37. RDMREG Command/Result Formats

Table 31-21 shows the definition of DRc encoding.

Table 31-21. Definition of DRc Encoding—Read

DRc[4:0]	Debug Register Definition	Mnemonic	Initial State	Page
0x00	Configuration/Status	CSR	0x0	p. 31-9
0x01–0x1F	Reserved	—	—	—

Command Sequence:

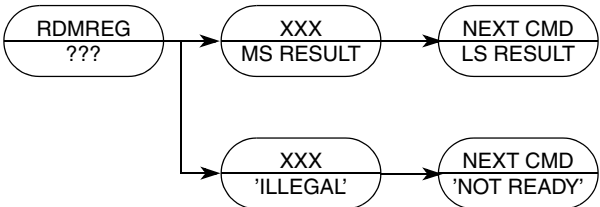


Figure 31-38. RDMREG Command Sequence

Operand Data:           None

Result Data:            The contents of the selected debug register are returned as a longword value.  
                          The data is returned most-significant word first.

31.5.3.3.12Write Debug Module Register (WDMREG)

The operand (longword) data is written to the specified debug module register. All 32 bits of the register are altered by the write. DSCLK must be inactive while the debug module register writes from the CPU accesses are performed using the WDEBUG instruction.

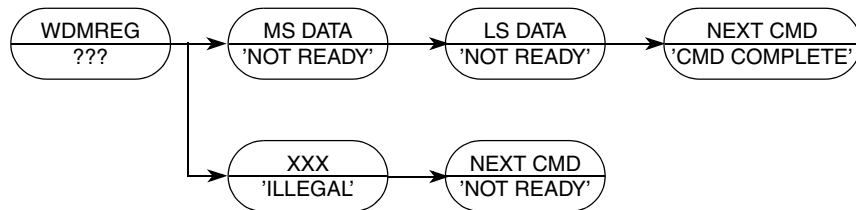
Command Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x2				0xC				100				DRc			
D[31:16]															
D[15:0]															

Figure 31-39. WDMREG BDM Command Format

Table 31-4 shows the definition of the DRc write encoding.

Command Sequence:



**Figure 31-40. WDMREG Command Sequence**

**Operand Data:** Longword data is written into the specified debug register. The data is supplied most-significant word first.

**Result Data:** Command complete status (0xFFFF) is returned when register write is complete.

## 31.6 Real-Time Debug Support

The ColdFire Family provides support debugging real-time applications. For these types of embedded systems, the processor must continue to operate during debug. The foundation of this area of debug support is that while the processor cannot be halted to allow debugging, the system can generally tolerate small intrusions into the real-time operation.

The debug module provides three types of breakpoints—PC with mask, operand address range, and data with mask. These breakpoints can be configured into one- or two-level triggers with the exact trigger response also programmable. The debug module programming model can be written from either the external development system using the debug serial interface or from the processor's supervisor programming model using the WDEBUG instruction. Only CSR is readable using the external development system.

### 31.6.1 Theory of Operation

Breakpoint hardware can be configured to respond to triggers in several ways. The response desired is programmed into TDR. As shown in Table 31-22, when a breakpoint is triggered, an indication (CSR[BSTAT]) is provided on the DDATA output port when it is not displaying captured processor status, operands, or branch addresses.

**Table 31-22. DDATA[3:0]/CSR[BSTAT] Breakpoint Response**

DDATA[3:0] <sup>1</sup>	CSR[BSTAT] <sup>1</sup>	Breakpoint Status
0000	0000	No breakpoints enabled
0010	0001	Waiting for level-1 breakpoint
0100	0010	Level-1 breakpoint triggered
1010	0101	Waiting for level-2 breakpoint
1100	0110	Level-2 breakpoint triggered

<sup>1</sup> Encodings not shown are reserved for future use.

The breakpoint status is also posted in the CSR. Note that CSR[BSTAT] is cleared by a CSR read when either a level-2 breakpoint is triggered or a level-1 breakpoint is triggered and a level-2 breakpoint is not enabled. Status is also cleared by writing to TDR.

BDM instructions use the appropriate registers to load and configure breakpoints. As the system operates, a breakpoint trigger generates the response defined in TDR.

PC breakpoints are treated in a precise manner—exception recognition and processing are initiated before the excepting instruction is executed. All other breakpoint events are recognized on the processor's local bus, but are made pending to the processor and sampled like other interrupt conditions. As a result, these interrupts are imprecise.

In systems that tolerate the processor being halted, a BDM-entry can be used. With TDR[TRC] = 01, a breakpoint trigger causes the core to halt (PST = 0xF).

If the processor core cannot be halted, the debug interrupt can be used. With this configuration, TDR[TRC] = 10, the breakpoint trigger becomes a debug interrupt to the processor, which is treated higher than the nonmaskable level-7 interrupt request. As with all interrupts, it is made pending until the processor reaches a sample point, which occurs once per instruction. Again, the hardware forces the PC breakpoint to occur before the targeted instruction executes. This is possible because the PC breakpoint is enabled when interrupt sampling occurs. For address and data breakpoints, reporting is considered imprecise because several instructions may execute after the triggering address or data is detected.

As soon as the debug interrupt is recognized, the processor aborts execution and initiates exception processing. This event is signaled externally by the assertion of a unique PST value (PST = 0xD) for multiple cycles. The core enters emulator mode when exception processing begins. After the standard 8-byte exception stack is created, the processor fetches a unique exception vector, 12, from the vector table.

Execution continues at the instruction address in the vector corresponding to the breakpoint triggered. All interrupts are ignored while the processor is in emulator mode. The debug interrupt handler can use supervisor instructions to save the necessary context such as the state of all program-visible registers into a reserved memory area.

When debug interrupt operations complete, the RTE instruction executes and the processor exits emulator mode. After the debug interrupt handler completes execution, the external development system can use BDM commands to read the reserved memory locations.

If a hardware breakpoint such as a PC trigger is left unmodified by the debug interrupt service routine, another debug interrupt is generated after the completion of the RTE instruction.

### 31.6.1.1 Emulator Mode

Emulator mode is used to facilitate non-intrusive emulator functionality. This mode can be entered in three different ways:

- Setting CSR[EMU] forces the processor into emulator mode. EMU is examined only if RESET is negated and the processor begins reset exception processing. It can be set while the processor is halted before reset exception processing begins. See [Section 31.5.1, “CPU Halt.”](#)
- A debug interrupt always puts the processor in emulation mode when debug interrupt exception processing begins.
- Setting CSR[TRC] forces the processor into emulation mode when trace exception processing begins.

While operating in emulation mode, the processor exhibits the following properties:

- All interrupts are ignored, including level-7 interrupts.
- If CSR[MAP] = 1, all caching of memory and the SRAM module are disabled. All memory accesses are forced into a specially mapped address space signaled by TT = 0x2, TM = 0x5 or 0x6. This includes stack frame writes and the vector fetch for the exception that forced entry into this mode.

The RTE instruction exits emulation mode. The processor status output port provides a unique encoding for emulator mode entry (0xD) and exit (0x7).

### 31.6.2 Concurrent BDM and Processor Operation

The debug module supports concurrent operation of both the processor and most BDM commands. BDM commands may be executed while the processor is running, except those following operations that access processor/memory registers:

- Read/write address and data registers
- Read/write control registers

For BDM commands that access memory, the debug module requests the processor's local bus. The processor responds by stalling the instruction fetch pipeline and waiting for current bus activity to complete before freeing the local bus for the debug module to perform its access. After the debug module bus cycle, the processor reclaims the bus.

Breakpoint registers must be carefully configured in a development system if the processor is executing. The debug module contains no hardware interlocks, so TDR should be disabled while breakpoint registers are loaded, after which TDR can be written to define the exact trigger. This prevents spurious breakpoint triggers.

Because there are no hardware interlocks in the debug unit, no BDM operations are allowed while the CPU is writing the debug's registers (DSCLK must be inactive).

Note that the debug module requires the use of the internal bus to perform BDM commands. In Revision A, if the processor is executing a tight loop that is contained within a single aligned longword, the processor may never grant the internal bus to the debug module, for example:

```
        align4
label1: nop
        bra.b label1
```

or

```
        align4
label2: bra.w label2
```

The processor grants the internal bus if these loops are forced across two longwords.

## 31.7 Processor Status, DDATA Definition

This section specifies the ColdFire processor and debug module's generation of the processor status (PST) and debug data (DDATA) output on an instruction basis. In general, the PST/DDATA output for an instruction is defined as follows:

$$\text{PST} = 0x1, \{ \text{PST} = [0x89B], \text{DDATA} = \text{operand} \}$$

where the {...} definition is optional operand information defined by the setting of the CSR.

The CSR provides capabilities to display operands based on reference type (read, write, or both). A PST value {0x8, 0x9, or 0xB} identifies the size and presence of valid data to follow on the DDATA output {1, 2, or 4 bytes}. Additionally, for certain change-of-flow branch instructions, CSR[BTB] provides the capability to display the target instruction address on the DDATA output {2, 3, or 4 bytes} using a PST value of {0x9, 0xA, or 0xB}.

### 31.7.1 User Instruction Set

Table 31-23 shows the PST/DDATA specification for user-mode instructions. Rn represents any {Dn, An} register. In this definition, the 'y' suffix generally denotes the source and 'x' denotes the destination operand. For a given instruction, the optional operand data is displayed only for those effective addresses referencing memory. The 'DD' nomenclature refers to the DDATA outputs.



**Table 31-23. PST/DDATA Specification for User-Mode Instructions**

Instruction	Operand Syntax	PST/DDATA
add.l	<ea>y,Rx	PST = 0x1, {PST = 0xB, DD = source operand}
add.l	Dy,<ea>x	PST = 0x1, {PST = 0xB, DD = source}, {PST = 0xB, DD = destination}
addi.l	#imm,Dx	PST = 0x1
addq.l	#imm,<ea>x	PST = 0x1, {PST = 0xB, DD = source}, {PST = 0xB, DD = destination}
addx.l	Dy,Dx	PST = 0x1
and.l	<ea>y,Dx	PST = 0x1, {PST = 0xB, DD = source operand}
and.l	Dy,<ea>x	PST = 0x1, {PST = 0xB, DD = source}, {PST = 0xB, DD = destination}
andi.l	#imm,Dx	PST = 0x1
asl.l	{Dy,#imm},Dx	PST = 0x1
asr.l	{Dy,#imm},Dx	PST = 0x1
bitrev.l	Dx	PST = 0x1
byterev.l	Dx	PST = 0x1
bcc.{b,w}		if taken, then PST = 0x5, else PST = 0x1
bchg	#imm,<ea>x	PST = 0x1, {PST = 0x8, DD = source}, {PST = 0x8, DD = destination}
bchg	Dy,<ea>x	PST = 0x1, {PST = 0x8, DD = source}, {PST = 0x8, DD = destination}
bclr	#imm,<ea>x	PST = 0x1, {PST = 0x8, DD = source}, {PST = 0x8, DD = destination}
bclr	Dy,<ea>x	PST = 0x1, {PST = 0x8, DD = source}, {PST = 0x8, DD = destination}
bra.{b,w}		PST = 0x5
bset	#imm,<ea>x	PST = 0x1, {PST = 0x8, DD = source}, {PST = 0x8, DD = destination}
bset	Dy,<ea>x	PST = 0x1, {PST = 0x8, DD = source}, {PST = 0x8, DD = destination}
bsr.{b,w}		PST = 0x5, {PST = 0xB, DD = destination operand}
btst	#imm,<ea>x	PST = 0x1, {PST = 0x8, DD = source operand}
btst	Dy,<ea>x	PST = 0x1, {PST = 0x8, DD = source operand}
clr.b	<ea>x	PST = 0x1, {PST = 0x8, DD = destination operand}
clr.l	<ea>x	PST = 0x1, {PST = 0xB, DD = destination operand}
clr.w	<ea>x	PST = 0x1, {PST = 0x9, DD = destination operand}
cmp.l	<ea>y,Rx	PST = 0x1, {PST = 0xB, DD = source operand}
cmpi.l	#imm,Dx	PST = 0x1
divs.l	<ea>y,Dx	PST = 0x1, {PST = 0xB, DD = source operand}
divs.w	<ea>y,Dx	PST = 0x1, {PST = 0x9, DD = source operand}
divu.l	<ea>y,Dx	PST = 0x1, {PST = 0xB, DD = source operand}
divu.w	<ea>y,Dx	PST = 0x1, {PST = 0x9, DD = source operand}

**Table 31-23. PST/DDATA Specification for User-Mode Instructions (Continued)**

Instruction	Operand Syntax	PST/DDATA
eor.l	Dy,<ea>x	PST = 0x1, {PST = 0xB, DD = source}, {PST = 0xB, DD = destination}
eori.l	#imm,Dx	PST = 0x1
ext.l	Dx	PST = 0x1
ext.w	Dx	PST = 0x1
extb.l	Dx	PST = 0x1
ff1.l	Dx	PST = 0x1
jmp	<ea>x	PST = 0x5, {PST = [0x9AB], DD = target address} <sup>1</sup>
jsr	<ea>x	PST = 0x5, {PST = [0x9AB], DD = target address}, {PST = 0xB, DD = destination operand} <sup>1</sup>
lea	<ea>y,Ax	PST = 0x1
link.w	Ay,#imm	PST = 0x1, {PST = 0xB, DD = destination operand}
lsl.l	{Dy,#imm},Dx	PST = 0x1
lsr.l	{Dy,#imm},Dx	PST = 0x1
move.b	<ea>y,<ea>x	PST = 0x1, {PST = 0x8, DD = source}, {PST = 0x8, DD = destination}
move.l	<ea>y,<ea>x	PST = 0x1, {PST = 0xB, DD = source}, {PST = 0xB, DD = destination}
move.w	<ea>y,<ea>x	PST = 0x1, {PST = 0x9, DD = source}, {PST = 0x9, DD = destination}
move.w	CCR,Dx	PST = 0x1
move.w	{Dy,#imm},CCR	PST = 0x1
movem.l	#list,<ea>x	PST = 0x1, {PST = 0xB, DD = destination},... <sup>2</sup>
movem.l	<ea>y,#list	PST = 0x1, {PST = 0xB, DD = source},... <sup>2</sup>
moveq	#imm,Dx	PST = 0x1
muls.l	<ea>y,Dx	PST = 0x1, {PST = 0xB, DD = source operand}
muls.w	<ea>y,Dx	PST = 0x1, {PST = 0x9, DD = source operand}
mulu.l	<ea>y,Dx	PST = 0x1, {PST = 0xB, DD = source operand}
mulu.w	<ea>y,Dx	PST = 0x1, {PST = 0x9, DD = source operand}
neg.l	Dx	PST = 0x1
negx.l	Dx	PST = 0x1
nop		PST = 0x1
not.l	Dx	PST = 0x1
or.l	<ea>y,Dx	PST = 0x1, {PST = 0xB, DD = source operand}
or.l	Dy,<ea>x	PST = 0x1, {PST = 0xB, DD = source}, {PST = 0xB, DD = destination}
ori.l	#imm,Dx	PST = 0x1
pea	<ea>y	PST = 0x1, {PST = 0xB, DD = destination operand}

**Table 31-23. PST/DDATA Specification for User-Mode Instructions (Continued)**

Instruction	Operand Syntax	PST/DDATA
pulse		PST = 0x4
rems.l	<ea>y,Dx:Dw	PST = 0x1, {PST = 0xB, DD = source operand}
remu.l	<ea>y,Dx:Dw	PST = 0x1, {PST = 0xB, DD = source operand}
rts		PST = 0x1, {PST = 0xB, DD = source operand}, PST = 0x5, {PST = [0x9AB], DD = target address}
scc	Dx	PST = 0x1
sub.l	<ea>y,Rx	PST = 0x1, {PST = 0xB, DD = source operand}
sub.l	Dy,<ea>x	PST = 0x1, {PST = 0xB, DD = source}, {PST = 0xB, DD = destination}
subi.l	#imm,Dx	PST = 0x1
subq.l	#imm,<ea>x	PST = 0x1, {PST = 0xB, DD = source}, {PST = 0xB, DD = destination}
subx.l	Dy,Dx	PST = 0x1
swap	Dx	PST = 0x1
trap	#imm	PST = 0x1 <sup>3</sup>
trapf		PST = 0x1
tst.b	<ea>x	PST = 0x1, {PST = 0x8, DD = source operand}
tst.l	<ea>x	PST = 0x1, {PST = 0xB, DD = source operand}
tst.w	<ea>x	PST = 0x1, {PST = 0x9, DD = source operand}
unlk	Ax	PST = 0x1, {PST = 0xB, DD = destination operand}
wddata.b	<ea>y	PST = 0x4, {PST = 0x8, DD = source operand}
wddata.l	<ea>y	PST = 0x4, {PST = 0xB, DD = source operand}
wddata.w	<ea>y	PST = 0x4, {PST = 0x9, DD = source operand}

<sup>1</sup> For JMP and JSR instructions, the optional target instruction address is displayed only for those effective address fields defining variant addressing modes. This includes the following <ea>x values: (An), (d16,An), (d8,An,Xi), (d8,PC,Xi).

<sup>2</sup> For Move Multiple instructions (MOVEM), the processor automatically generates line-sized transfers if the operand address reaches a 0-modulo-16 boundary and there are four or more registers to be transferred. For these line-sized transfers, the operand data is never captured nor displayed, regardless of the CSR value. The automatic line-sized burst transfers are provided to maximize performance during these sequential memory access operations.

<sup>3</sup> During normal exception processing, the PST output is driven to a 0xC indicating the exception processing state. The exception stack write operands, as well as the vector read and target address of the exception handler may also be displayed.

```
Exception ProcessingPST = 0xC,{PST = 0xB,DD = destination},// stack frame
                    {PST = 0xB,DD = destination},// stack frame
                    {PST = 0xB,DD = source},// vector read
                    PST = 0x5,{PST = [0x9AB],DD = target}// handler PC
```

The PST/DDATA specification for the reset exception is shown below:

```
Exception Processing PST = 0xC,
      PST = 0x5, {PST = [0x9AB], DD = target} // handler PC
```

The initial references at address 0 and 4 are never captured nor displayed since these accesses are treated as instruction fetches.

For all types of exception processing, the PST = 0xC value is driven at all times, unless the PST output is needed for one of the optional marker values or for the taken branch indicator (0x5).

Table 31-24 shows the PST/DDATA specification for multiply-accumulate instructions.

**Table 31-24. PST/DDATA Specification for MAC Instructions**

Instruction	Operand Syntax	PST/DDATA
mac.l	Ry,Rx,Accx	PST = 0x1
mac.l	Ry,Rx,<ea>,Rw,Accx	PST = 0x1, {PST = 0xB, DD = source operand}
mac.w	Ry,Rx,Accx	PST = 0x1
mac.w	Ry,Rx,<ea>,Rw,Accx	PST = 0x1, {PST = 0xB, DD = source operand}
move.l	<ea>y,Accx	PST = 0x1
move.l	Accy,Accx	PST = 0x1
move.l	<ea>y,MACR	PST = 0x1
move.l	<ea>y,MASK	PST = 0x1
move.l	<ea>y,Accext01	PST = 0x1
move.l	<ea>y,Accext23	PST = 0x1
move.l	Accy,Rx	PST = 0x1
move.l	MACSR,CCR	PST = 0x1
move.l	MACSR,Rx	PST = 0x1
move.l	MASK,Rx	PST = 0x1
move.l	Accext01,Rx	PST = 0x1
move.l	Accext23,Rx	PST = 0x1
msac.l	Ry,Rx,Accx	PST = 0x1
msac.l	Ry,Rx,<ea>,Rw,Accx	PST = 0x1, {PST = 0xB, DD = source operand}
msac.w	Ry,Rx,Accx	PST = 0x1
msac.w	Ry,Rx,<ea>,Rw,Accx	PST = 0x1, {PST = 0xB, DD = source operand}

## 31.7.2 Supervisor Instruction Set

The supervisor instruction set has complete access to the user mode instructions plus the opcodes shown below. The PST/DDATA specification for these opcodes is shown in Table 31-25.

**Table 31-25. PST/DDATA Specification for Supervisor-Mode Instructions**

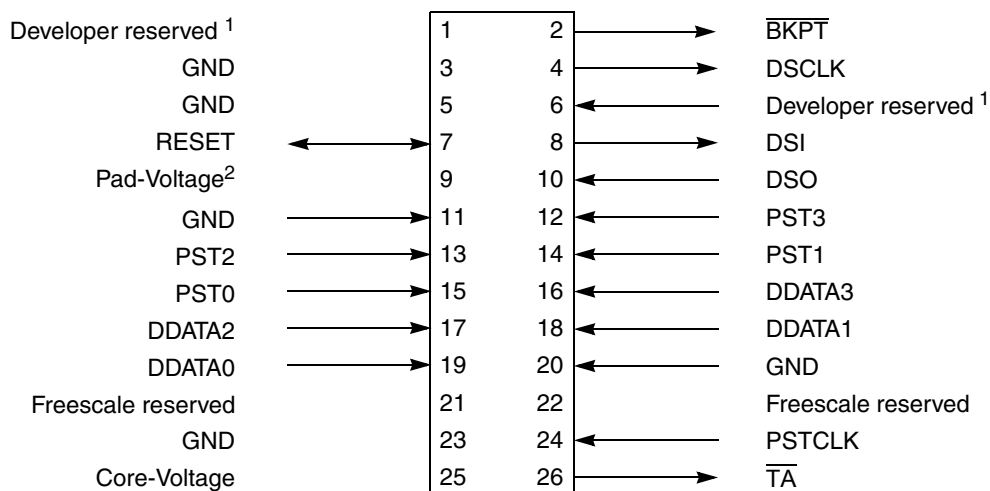
Instruction	Operand Syntax	PST/DDATA
cpushl		PST = 0x1
halt		PST = 0x1, PST = 0xF
move.w	SR,Dx	PST = 0x1
move.w	{Dy,#imm},SR	PST = 0x1, {PST = 0x3}
movec	Ry,Rc	PST = 0x1
rte		PST = 0x7, {PST = 0xB, DD = source operand}, {PST = 3},{ PST =0xB, DD =source operand}, PST = 0x5, {[PST = 0x9AB], DD = target address}
stldsr.w	#imm	PST = 0x1, {PST = 0xA, DD = destination operand, PST = 0x3}
stop	#imm	PST = 0x1, PST = 0xE
wdebug	<ea>y	PST = 0x1, {PST = 0xB, DD = source, PST = 0xB, DD = source}

The move-to-SR, STLDsr, and RTE instructions include an optional PST = 0x3 value, indicating an entry into user mode. Additionally, if the execution of a RTE instruction returns the processor to emulator mode, a multiple-cycle status of 0xD is signaled.

Similar to the exception processing mode, the stopped state (PST = 0xE) and the halted state (PST = 0xF) display this status throughout the entire time the ColdFire processor is in the given mode.

## 31.8 Freescale-Recommended BDM Pinout

The ColdFire BDM connector, [Figure 31-41](#), is a 26-pin Berg connector arranged 2×13.



<sup>1</sup>Pins reserved for BDM developer use.

<sup>2</sup>Supplied by target

**Figure 31-41. Recommended BDM Connector**

# Chapter 32

## IEEE 1149.1 Test Access Port (JTAG)

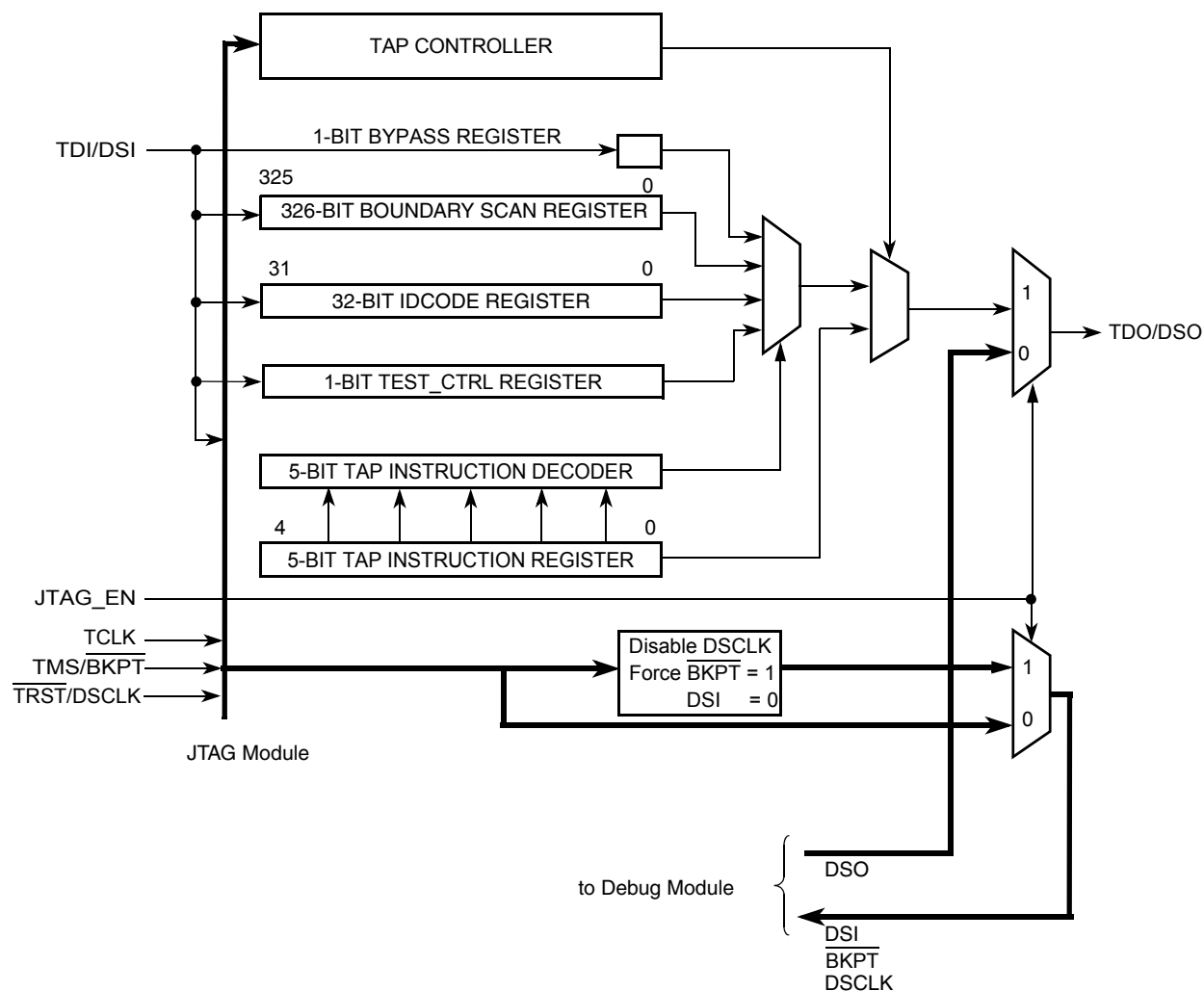
### 32.1 Introduction

The Joint Test Action Group, or JTAG, is a dedicated user-accessible test logic, that complies with the IEEE 1149.1 standard for boundary-scan testability, to help with system diagnostic and manufacturing testing.

This architecture provides access to all data and chip control pins from the board-edge connector through the standard four-pin test access port (TAP) and the JTAG reset pin,  $\overline{\text{TRST}}$ .

#### 32.1.1 Block Diagram

[Figure 32-1](#) shows the block diagram of the JTAG module.



### Figure 32-1. JTAG Block Diagram

### 32.1.2 Features

The basic features of the JTAG module are the following:

- Performs boundary-scan operations to test circuit board electrical continuity
- Bypasses instruction to reduce the shift register path to a single cell
- Sets chip output pins to safety states while executing the bypass instruction
- Samples the system pins during operation and transparently shift out the result
- Selects between JTAG TAP controller and Background Debug Module (BDM) using a dedicated JTAG EN pin



### 32.1.3 Modes of Operation

The JTAG\_EN pin can select between the following modes of operation:

- JTAG mode
- BDM - background debug mode (For more information, refer to [Section 31.5](#), “Background Debug Mode (BDM)).”

## 32.2 External Signal Description

The JTAG module has five input and one output external signals, as described in [Table 32-1](#).

**Table 32-1. Signal Properties**

Name	Direction	Function	Reset State	Pull up
JTAG_EN	Input	JTAG/BDM selector input	—	—
TCLK	Input	JTAG Test clock input	—	Active
TMS/BKPT	Input	JTAG Test mode select / BDM Breakpoint	—	Active
TDI/DSI	Input	JTAG Test data input / BDM Development serial input	—	Active
TRST/DSCLK	Input	JTAG Test reset input / BDM Development serial clock	—	Active
TDO/DSO	Output	JTAG Test data output / BDM Development serial output	Hi-Z / 0	—

### 32.2.1 JTAG Enable (JTAG\_EN)

The JTAG\_EN pin selects between the Debug module and JTAG. If JTAG\_EN is low, the Debug module is selected; if it is high, the JTAG is selected. [Table 32-2](#) summarizes the pin function selected depending upon JTAG\_EN logic state.

**Table 32-2. Pin Function Selected**

	JTAG_EN = 0	JTAG_EN = 1	Pin Name
Module selected	BDM	JTAG	—
Pin Function	— BKPT DSI DSO DSCLK	TCLK TMS TDI TDO TRST	TCLK BKPT DSI DSO DSCLK

When one module is selected, the inputs into the other module are disabled or forced to a known logic level as shown in [Table 32-3](#), in order to disable the corresponding module.

**Table 32-3. Signal State to the Disable Module**

	JTAG_EN = 0	JTAG_EN = 1
Disabling JTAG	$\overline{\text{TRST}} = 0$ TMS = 1	—
Disabling BDM	—	Disable DSCLK DSI = 0 $\overline{\text{BKPT}} = 1$

**NOTE**

The JTAG\_EN does not support dynamic switching between JTAG and BDM modes.

**32.2.2 Test Clock Input (TCLK)**

The TCLK pin is a dedicated JTAG clock input to synchronize the test logic. Pulses on TCLK shift data and instructions into the TDI pin on the rising edge and out of the TDO pin on the falling edge. TCLK is independent of the processor clock. The TCLK pin has an internal pull-up resistor and holding TCLK high or low for an indefinite period does not cause JTAG test logic to lose state information.

**32.2.3 Test Mode Select/Breakpoint (TMS/ $\overline{\text{BKPT}}$ )**

The TMS pin is the test mode select input that sequences the TAP state machine. TMS is sampled on the rising edge of TCLK. The TMS pin has an internal pull-up resistor.

The  $\overline{\text{BKPT}}$  pin is used to request an external breakpoint. Assertion of  $\overline{\text{BKPT}}$  puts the processor into a halted state after the current instruction completes.

**32.2.4 Test Data Input/Development Serial Input (TDI/DSI)**

The TDI pin receives serial test and data, which is sampled on the rising edge of TCLK. Register values are shifted in least significant bit (lsb) first. The TDI pin has an internal pull-up resistor.

The DSI pin provides data input for the debug module serial communication port.

**32.2.5 Test Reset/Development Serial Clock ( $\overline{\text{TRST}}$ /DSCLK)**

The  $\overline{\text{TRST}}$  pin is an active low asynchronous reset input with an internal pull-up resistor that forces the TAP controller to the test-logic-reset state.

The DSCLK pin clocks the serial communication port to the debug module. Maximum frequency is 1/5 the processor clock speed. At the rising edge of DSCLK, the data input on DSI is sampled and DSO changes state.

## 32.2.6 Test Data Output/Development Serial Output (TDO/DSO)

The TDO pin is the lsb-first data output. Data is clocked out of TDO on the falling edge of TCLK. TDO is tri-stateable and is actively driven in the shift-IR and shift-DR controller states.

The DSO pin provides serial output data in BDM mode.

## 32.3 Memory Map/Register Definition

The JTAG module registers are not memory mapped and are only accessible through the TDO/DSO pin.

### 32.3.1 Register Descriptions

All registers are shift-in and parallel load.

#### 32.3.1.1 Instruction Shift Register (IR)

The JTAG module uses a 4-bit shift register with no parity. The IR transfers its value to a parallel hold register and applies an instruction on the falling edge of TCLK when the TAP state machine is in the update-IR state. To load an instruction into the shift portion of the IR, place the serial data on the TDI pin before each rising edge of TCLK. The msb of the IR is the bit closest to the TDI pin, and the lsb is the bit closest to the TDO pin.

	3	2	1	0
R	0	1	0	1
W	Instruction Code			
Reset	0	0	0	1

**Figure 32-2. 4-Bit Instruction Register (IR)**

#### 32.3.1.2 IDCODE Register

The IDCODE is a read-only register; its value is chip dependent. For more information, see [Section 32.4.3.2, “IDCODE Instruction.”](#)

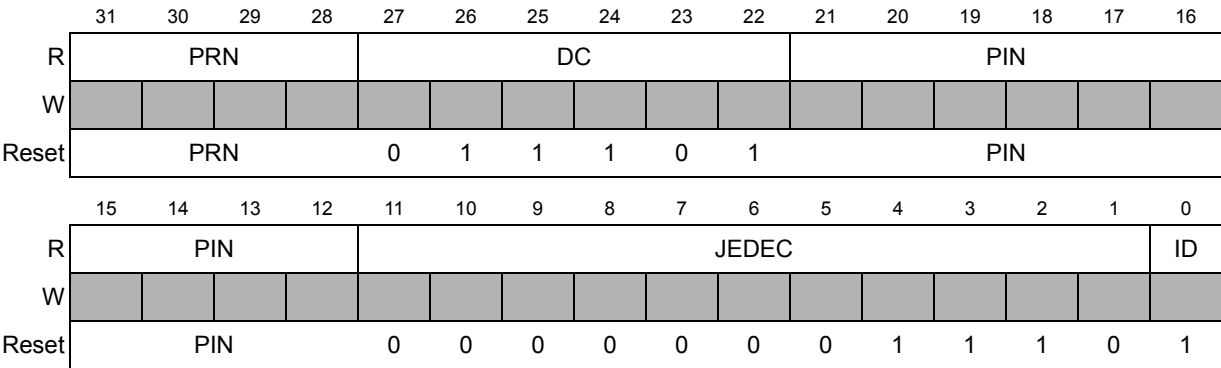


Table 32-4. IDCODE Field Descriptions

Bits	Name	Description
31–28	PRN	Part revision number. Indicate the revision number of the device.
27–22	DC	Freescale Design Center number.
21–12	PIN	Part identification number. Indicate the device number.
11–1	JEDEC	Joint Electron Device Engineering Council ID bits. Indicate the reduced JEDEC ID for Freescale.
0	ID	IDCODE register ID. This bit is set to 1 to identify the register as the IDCODE register and not the bypass register according to the IEEE standard 1149.1.

32.3.1.3 Bypass Register

The bypass register is a single-bit shift register path from TDI to TDO when the BYPASS, CLAMP, or HIGHZ instructions are selected. 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.

32.3.1.4 TEST\_CTRL Register

The TEST\_CTRL register is a 1-bit shift register path from TDI to TDO when the ENABLE\_TEST\_CTRL instruction is selected. The TEST\_CTRL transfers its value to a parallel hold register on the rising edge of TCLK when the TAP state machine is in the Update-DR state. The DSE bit selects the drive strength used in JTAG mode.

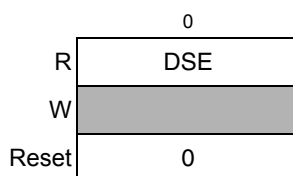


Figure 32-4. 1-Bit TEST\_CTRL Register

### 32.3.1.5 Boundary Scan Register

The boundary scan register is connected between TDI and TDO when the EXTEST, SAMPLE or SAMPLE/PRELOAD instruction is selected. It captures input pin data, forces fixed values on output pins, and selects a logic value and direction for bidirectional pins or high impedance for tri-stated pins.

The boundary scan register contains bits for bonded-out and non bonded-out signals excluding JTAG signals, analog signals, power supplies, compliance enable pins, device configuration pins, and clock signals.

## 32.4 Functional Description

### 32.4.1 JTAG Module

The JTAG module consists of a TAP controller state machine, which is responsible for generating all control signals that execute the JTAG instructions and read/write data registers.

### 32.4.2 TAP Controller

The TAP controller is a state machine that changes state based on the sequence of logical values on the TMS pin. [Figure 32-5](#) 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 TCLK signal.

Asserting the  $\overline{\text{TRST}}$  signal asynchronously resets the TAP controller to the test-logic-reset state. As [Figure 32-5](#) shows, holding TMS at logic 1 while clocking TCLK through at least five rising edges also causes the state machine to enter the test-logic-reset state, whatever the initial state.

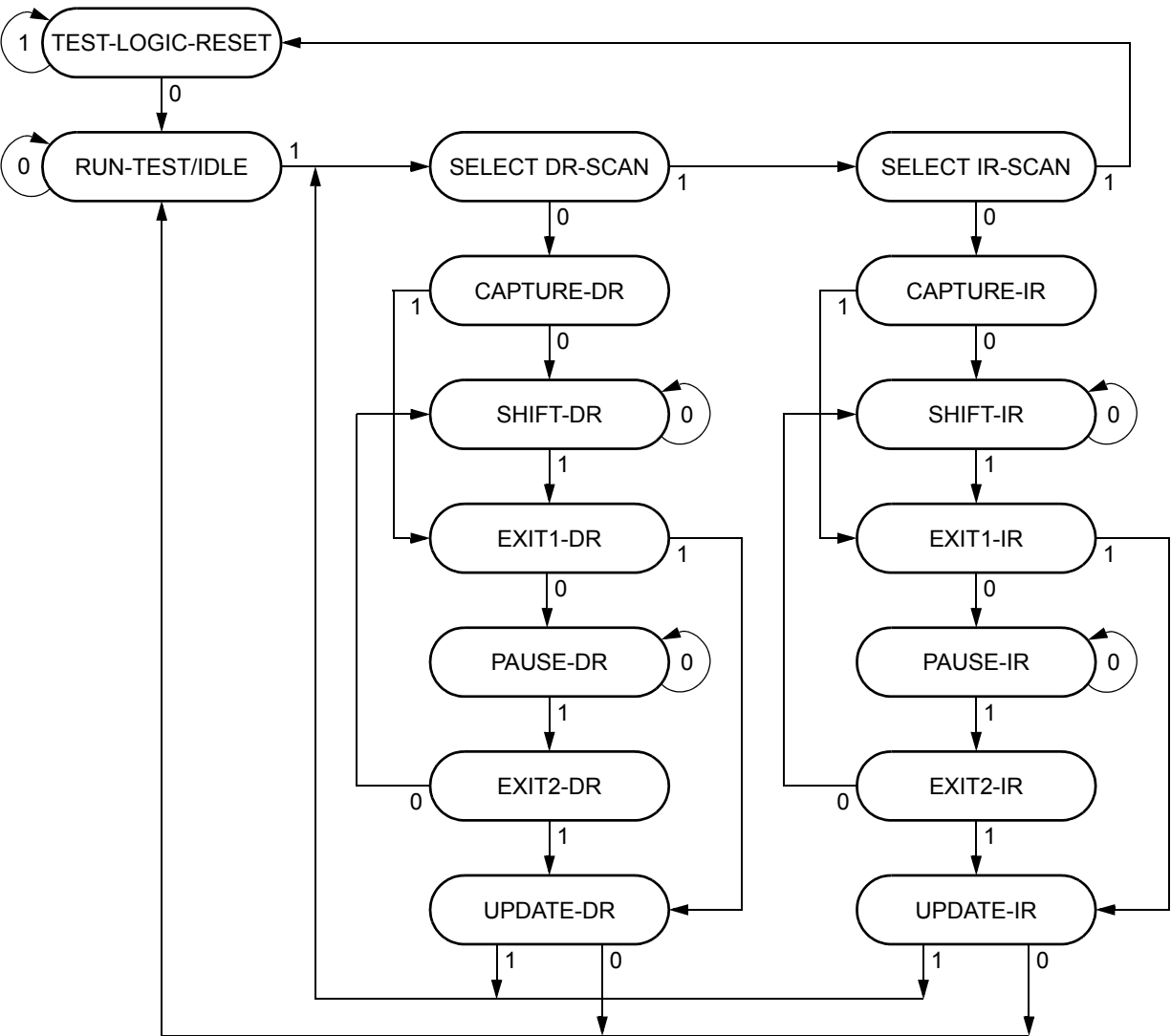


Figure 32-5. TAP Controller State Machine Flow

32.4.3 JTAG Instructions

The below table describes public and private instructions.

Table 32-5. JTAG Instructions

Instruction	IR[3:0]	Instruction Summary
EXTEST	0000	Selects boundary scan register while applying fixed values to output pins and asserting functional reset
IDCODE	0001	Selects IDCODE register for shift
SAMPLE/PRELOAD	0010	Selects boundary scan register for shifting, sampling, and preloading without disturbing functional operation

**Table 32-5. JTAG Instructions (Continued)**

Instruction	IR[3:0]	Instruction Summary
ENABLE_TEST_CTRL	0110	Selects TEST_CTRL register
HIGHZ	1001	Selects bypass register while tri-stating all output pins and asserting functional reset
CLAMP	1100	Selects bypass while applying fixed values to output pins and asserting functional reset
BYPASS	1111	Selects bypass register for data operations
Reserved	all others	Decoded to select bypass register <sup>1</sup>
EXTEST	0000	Selects boundary scan register while applying fixed values to output pins and asserting functional reset

<sup>1</sup>Freescall reserves the right to change the decoding of the unused opcodes in the future.

### 32.4.3.1 EXTEST Instruction

The external test (EXTEST) instruction selects the boundary scan register. It forces all output pins and bidirectional pins configured as outputs to the values preloaded with the SAMPLE/PRELOAD instruction and held in the boundary scan update registers. EXTEST can also configure the direction of bidirectional pins and establish high-impedance states on some pins. EXTEST asserts internal reset for the MCU system logic to force a predictable internal state while performing external boundary scan operations.

### 32.4.3.2 IDCODE Instruction

The IDCODE instruction selects the 32-bit IDCODE register for connection as a shift path between the TDI and TDO pin. This instruction allows interrogation of the MCU to determine its version number and other part identification data. The shift register lsb is forced to logic 1 on the rising edge of TCLK following entry into the capture-DR state. Therefore, the first bit to be shifted out after selecting the IDCODE register is always a logic 1. The remaining 31 bits are also forced to fixed values on the rising edge of TCLK following entry into the capture-DR state.

IDCODE is the default instruction placed into the instruction register when the TAP resets. Thus, after a TAP reset, the IDCODE register is selected automatically.

### 32.4.3.3 SAMPLE/PRELOAD Instruction

The SAMPLE/PRELOAD instruction has two functions:

- **SAMPLE** - obtain a sample of the system data and control signals present at the MCU input pins and just before the boundary scan cell at the output pins. This sampling occurs on the rising edge of TCLK in the capture-DR state when the IR contains the \$2 opcode. The sampled data is accessible by shifting it through the boundary scan register to the TDO

output by using the shift-DR state. Both the data capture and the shift operation are transparent to system operation.

### NOTE

External synchronization is required to achieve meaningful results because there is no internal synchronization between TCLK and the system clock.

- **PRELOAD** - initialize the boundary scan register update cells before selecting EXTEST or CLAMP. This is achieved by ignoring the data shifting out on the TDO pin and shifting in initialization data. The update-DR state and the falling edge of TCLK can then transfer this data to the update cells. The data is applied to the external output pins by the EXTEST or CLAMP instruction.

#### 32.4.3.4 ENABLE\_TEST\_CTRL Instruction

The ENABLE\_TEST\_CTRL instruction selects a 3-bit shift register (TEST\_CTRL) for connection as a shift path between the TDI and TDO pin. When the user transitions the TAP controller to the UPDATE\_DR state, the register transfers its value to a parallel hold register.

#### 32.4.3.5 HIGHZ Instruction

The HIGHZ instruction eliminates the need to backdrive the output pins during circuit-board testing. HIGHZ turns off all output drivers, including the 2-state drivers, and selects the bypass register. HIGHZ also asserts internal reset for the MCU system logic to force a predictable internal state.

#### 32.4.3.6 CLAMP Instruction

The CLAMP instruction selects the 1-bit bypass register and asserts internal reset while simultaneously forcing all output pins and bidirectional pins configured as outputs to the fixed values that are preloaded and held in the boundary scan update register. 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.

#### 32.4.3.7 BYPASS Instruction

The BYPASS instruction selects the bypass register, creating a single-bit shift register path from the TDI pin to the TDO pin. BYPASS enhances test efficiency by reducing the overall shift path when a device other than the ColdFire processor is the device under test on a board design with multiple chips on the overall boundary scan chain. The shift register lsb is forced to logic 0 on the rising edge of TCLK after entry into the capture-DR state. Therefore, the first bit shifted out after selecting the bypass register is always logic 0. This differentiates parts that support an IDCODE register from parts that support only the bypass register.



## 32.5 Initialization/Application Information

### 32.5.1 Restrictions

The test logic is a static logic design, and TCLK can be stopped in either a high or low state without loss of data. However, the system clock is not synchronized to TCLK internally. Any mixed operation using both the test logic and the system functional logic requires external synchronization.

Using the EXTEST instruction requires a circuit-board test environment that avoids device-destructive configurations in which MCU output drivers are enabled into actively driven networks.

Low-power stop mode considerations:

- The TAP controller must be in the test-logic-reset state to either enter or remain in the low-power stop mode. Leaving the test-logic-reset state negates the ability to achieve low-power, but does not otherwise affect device functionality.
- The TCLK input is not blocked in low-power stop mode. To consume minimal power, the TCLK input should be externally connected to  $V_{DD}$ .
- The TMS, TDI, and  $\overline{TRST}$  pins include on-chip pull-up resistors. For minimal power consumption in low-power stop mode, these three pins should be either connected to  $V_{DD}$  or left unconnected.

### 32.5.2 Nonscan Chain Operation

Keeping the TAP controller in the test-logic-reset state ensures that the scan chain test logic is transparent to the system logic. It is recommended that TMS, TDI, TCLK, and  $\overline{TRST}$  be pulled up.  $\overline{TRST}$  could be connected to ground. However, since there is a pull-up on  $\overline{TRST}$ , some amount of current results. The internal power-on reset input initializes the TAP controller to the test-logic-reset state on power-up without asserting  $\overline{TRST}$ .



# Appendix A

## Register Memory Map Quick Reference

### A.1 Register Memory Map

Table A-1 summarizes the address, name, and byte assignment for registers within the MCF5275 CPU space. Table A-2 lists an overview of the memory map for the on-chip modules. Each module is detailed in Table A-3 through Table A-24. Please refer to Section 11.2.1.1, “Internal Peripheral System Base Address Register (IPSBAR),” for determining the value of IPSBAR.

**Table A-1. CPU Space Register Memory Map**

Address	Name	Mnemonic	Size
CPU @ 0x002	Cache Control Register	CACR	32
CPU @ 0x004	Access Control Register 0	ACR0	32
CPU @ 0x005	Access Control Register 1	ACR1	32
CPU @ 0x800	Other Stack Pointer	OTHER_A7	32
CPU @ 0x801	Vector Base Register	VBR	32
CPU @ 0x804	MAC Status Register	MACSR	8
CPU @ 0x805	MAC Mask Register	MASK	16
CPU @ 0x806	MAC Accumulator 0	ACC0	16
CPU @ 0x807	MAC Accumulator 0, 1 extension bytes	ACCext01	16
CPU @ 0x808	MAC Accumulator 2, 3 extension bytes	ACCext23	16
CPU @ 0x809	MAC Accumulator 1	ACC1	16
CPU @ 0x80A	MAC Accumulator 2	ACC2	16
CPU @ 0x80B	MAC Accumulator 3	ACC3	16
CPU @ 0x80E	Status Register	SR	16
CPU @ 0x80F	Program Counter	PC	32
CPU @ 0xC05	RAM Base Address Register	RAMBAR	32

**Table A-2. MCF5275 Memory Map**

Address	Module Name	Size	Module Map
IPSBAR + 0x00 0000	System Control Module (SCM)	64 bytes	<a href="#">Table A-3</a>
IPSBAR + 0x00 0040	DDR SDRAMC	64 bytes	<a href="#">Table A-4</a>
IPSBAR + 0x00 0080	EIM (Chip Selects)	128 bytes	<a href="#">Table A-5</a>

**Table A-2. MCF5275 Memory Map (Continued)**

Address	Module Name	Size	Module Map
IPSBAR + 0x00 0100	DMA (Channel 0)	64 bytes	<a href="#">Table A-6</a>
IPSBAR + 0x00 0110	DMA (Channel 1)	64 bytes	
IPSBAR + 0x00 0120	DMA (Channel 2)	64 bytes	
IPSBAR + 0x00 0130	DMA (Channel 3)	64 bytes	
IPSBAR + 0x00 0200	UART0	64 bytes	<a href="#">Table A-7</a>
IPSBAR + 0x00 0240	UART1	64 bytes	
IPSBAR + 0x00 0280	UART2	64 bytes	
IPSBAR + 0x00 02C0	Reserved	64 bytes	—
IPSBAR + 0x00 0300	I <sup>2</sup> C	64 bytes	<a href="#">Table A-8</a>
IPSBAR + 0x00 0340	QSPI	64 bytes	<a href="#">Table A-9</a>
IPSBAR + 0x00 0380	Reserved	128 bytes	—
IPSBAR + 0x00 0400	DMA Timer0	64 bytes	<a href="#">Table A-10</a>
IPSBAR + 0x00 0440	DMA Timer1	64 bytes	
IPSBAR + 0x00 0480	DMA Timer2	64 bytes	
IPSBAR + 0x00 04C0	DMA Timer3	64 bytes	
IPSBAR + 0x00 0500	Reserved	1792 bytes	—
IPSBAR + 0x00 0C00	INTC0	256 bytes	<a href="#">Table A-11</a>
IPSBAR + 0x00 0D00	INTC1	256 bytes	
IPSBAR + 0x00 0E00	Reserved	256 bytes	—
IPSBAR + 0x00 0F00	Global Interrupt Ack Cycles	256 bytes	
IPSBAR + 0x00 1000	Fast Ethernet Controller 0 - Registers and MIB RAM	1 Kbyte	<a href="#">Table A-12</a>
IPSBAR + 0x00 1400	Fast Ethernet Controller 0 - FIFO Memory	1 Kbyte	<a href="#">Table A-13</a>
IPSBAR + 0x00 1800	Fast Ethernet Controller 1 - Registers and MIB RAM	1 Kbyte	<a href="#">Table A-12</a>
IPSBAR + 0x00 1C00	Fast Ethernet Controller 1 - FIFO Memory	1 Kbyte	<a href="#">Table A-13</a>
IPSBAR + 0x00 2000	Reserved	1016 Kbytes	—
IPSBAR + 0x10 0000	GPIO	64 Kbytes	<a href="#">Table A-14</a>
IPSBAR + 0x11 0000	CCM	64 Kbytes	<a href="#">Table A-15</a>
IPSBAR + 0x12 0000	Clocks (PLL)	64 Kbytes	<a href="#">Table A-16</a>
IPSBAR + 0x13 0000	Edge Port	64 Kbytes	<a href="#">Table A-17</a>
IPSBAR + 0x14 0000	Watchdog Timer	64 Kbytes	<a href="#">Table A-18</a>

**Table A-2. MCF5275 Memory Map (Continued)**

Address	Module Name	Size	Module Map
IPSBAR + 0x15 0000	PIT0	64 Kbytes	Table A-19
IPSBAR + 0x16 0000	PIT1	64 Kbytes	
IPSBAR + 0x17 0000	PIT2	64 Kbytes	
IPSBAR + 0x18 0000	PIT3	64 Kbytes	
IPSBAR + 0x19 0000	MDHA	64 Kbytes	Table A-20
IPSBAR + 0x1A 0000	RNG	64 Kbytes	Table A-21
IPSBAR + 0x1B 0000	SKHA	64 Kbytes	Table A-22
IPSBAR + 0x1C 0000	USB	64 Kbytes + 32	
IPSBAR + 0x1D 0000	PWM0–3	64 Kbytes - 32	
IPSBAR + 0x1E 0000	Reserved	~1022 Mbytes	—

**Table A-3. SCM Memory Map****Base Address: IPSBAR**

Offset	Page	Register Description	[31:24]	[23:16]	[15:8]	[7:0]
0x00_0000	11-3	Internal Peripheral System Base Address Register	IPSBAR			
0x00_0004	—					
0x00_0008	11-4	Memory Base Address Register	RAMBAR			
0x00_000C	—					
0x00_0010	11-6 11-7 8-2 11-8	Core Reset Status Register Core Watchdog Control Register Low Power Interrupt Control Register Core Watchdog Service Register	CRSR	CWCR	LPICR <sup>1</sup>	CWSR
0x00_0018	—					
0x00_001C	11-11	Bus Master Park Register	MPARK			
0x00_0020	11-15	Master Privilege Register	MPR			
0x00_0024	11-15	Peripheral Access Control Registers (0–3)	PACR0	PACR1	PACR2	PACR3
0x00_0028	11-15	Peripheral Access Control Registers (4–6)	PACR4		PACR5	PACR6

**Table A-3. SCM Memory Map (Continued)****Base Address: IPSBAR**

Offset	Page	Register Description	[31:24]	[23:16]	[15:8]	[7:0]
0x00_002C	<a href="#">11-15</a>	Peripheral Access Control Registers (7–8)	PACR7		PACR8	
0x00_0030	<a href="#">11-17</a>	Grouped Peripheral Access Control Register	GPACR			

<sup>1</sup> The LPICR is described in [Chapter 8, “Power Management.”](#)**Table A-4. DDR SDRAMC Memory Map****Base Address: IPSBAR**

Offset	Page	Register Description	[31:24]	[23:16]	[15:8]	[7:0]
0x00_0040	<a href="#">17-14</a>	SDRAM Mode/Extended Mode Register	SDMR			
0x00_0044	<a href="#">17-15</a>	SDRAM Control Register	SDCR			
0x00_0048	<a href="#">17-17</a>	SDRAM Configuration Register A	SDCFG A			
0x00_004C	<a href="#">17-19</a>	SDRAM Configuration Register B	SDCFG B			
0x00_0050	<a href="#">17-20</a>	SDRAM Base Address Register 0	SDBAR0			
0x00_0054	<a href="#">17-20</a>	SDRAM Base Address Mask Register 0	SDBMR0			
0x00_0058	<a href="#">17-20</a>	SDRAM Base Address Register 1	SDBAR1			
0x00_005C	<a href="#">17-20</a>	SDRAM Base Address Mask Register 1	SDBMR1			

**Table A-5. EIM (Chip Selects) Memory Map****Base Address: IPSBAR**

Offset	Page	Register Description	[31:24]	[23:16]	[15:8]	[7:0]
0x00_0080	<a href="#">15-7</a>	Chip Select Address Register—Bank 0	CSAR0			
0x00_0084	<a href="#">15-8</a>	Chip Select Mask Register—Bank 0	CSMR0			
0x00_0088	<a href="#">15-9</a>	Chip Select Control Register—Bank 0			CSCR0	
0x00_008C	<a href="#">15-7</a>	Chip Select Address Register—Bank 1	CSAR1			
0x00_0090	<a href="#">15-8</a>	Chip Select Mask Register—Bank 1	CSMR1			
0x00_0094	<a href="#">15-9</a>	Chip Select Control Register—Bank 1			CSCR1	
0x00_0098	<a href="#">15-7</a>	Chip Select Address Register—Bank 2	CSAR2			

**Table A-5. EIM (Chip Selects) Memory Map (Continued)****Base Address: IPSBAR**

Offset	Page	Register Description	[31:24]	[23:16]	[15:8]	[7:0]
0x00_009C	15-8	Chip Select Mask Register—Bank 2	CSMR2			
0x00_00A0	15-9	Chip Select Control Register—Bank 2			CSCR2	
0x00_00A4	15-7	Chip Select Address Register—Bank 3	CSAR3			
0x00_00A8	15-8	Chip Select Mask Register—Bank 3	CSMR3			
0x00_00AC	15-9	Chip Select Control Register—Bank 3			CSCR3	
0x00_00B0	15-7	Chip Select Address Register—Bank 4	CSAR4			
0x00_00B4	15-8	Chip Select Mask Register—Bank 4	CSMR4			
0x00_00B8	15-9	Chip Select Control Register—Bank 4			CSCR4	
0x00_00BC	15-7	Chip Select Address Register—Bank 5	CSAR5			
0x00_00C0	15-8	Chip Select Mask Register—Bank 5	CSMR5			
0x00_00C4	15-9	Chip Select Control Register—Bank 5			CSCR5	
0x00_00C8	15-7	Chip Select Address Register—Bank 6	CSAR6			
0x00_00CC	15-8	Chip Select Mask Register—Bank 6	CSMR6			
0x00_00D0	15-9	Chip Select Control Register—Bank 6			CSCR6	
0x00_00D4	15-7	Chip Select Address Register—Bank 7	CSAR7			
0x00_00D8	15-8	Chip Select Mask Register—Bank 7	CSMR7			
0x00_00DC	15-9	Chip Select Control Register—Bank 7			CSCR7	

**Table A-6. DMA Memory Map****Base Address:****DMA0: IPSBAR + 0x00\_0100****DMA1: IPSBAR + 0x00\_0110****DMA2: IPSBAR + 0x00\_0120****DMA3: IPSBAR + 0x00\_0130**

Offset	Page	Register Description	[31:24]	[23:16]	[15:8]	[7:0]
0x0	18-7	Source Address Register <i>n</i>	SAR <sub><i>n</i></sub>			
0x4	18-8	Destination Address Register <i>n</i>	DAR <sub><i>n</i></sub>			

**Table A-6. DMA Memory Map (Continued)****Base Address:****DMA0:** IPSBAR + 0x00\_0100**DMA1:** IPSBAR + 0x00\_0110**DMA2:** IPSBAR + 0x00\_0120**DMA3:** IPSBAR + 0x00\_0130

Offset	Page	Register Description	[31:24]	[23:16]	[15:8]	[7:0]
0x8	18-9 18-8	DMA Status Register <i>n</i> Byte Count Register <i>n</i>	DSR <i>n</i>	BCR <i>n</i>		
0xC	18-10	DMA Control Register <i>n</i>	DCR <i>n</i>			

**Table A-7. UART Memory Map****Base Address:****UART0:** IPSBAR + 0x00\_0200**UART1:** IPSBAR + 0x00\_0240**UART2:** IPSBAR + 0x00\_0280

Offset	Page	Register Description	[31:24]	[23:16]	[15:8]	[7:0]
0x00	27-5 27-7	UART Mode Registers <sup>1</sup>	UMR1 <sub><i>n</i></sub> UMR2 <sub><i>n</i></sub>			
0x04	27-8 27-10	(Read) UART Status Register (Write) UART Clock Select Register <sup>1</sup>	USR <sub><i>n</i></sub> UCSR <sub><i>n</i></sub>			
0x08	27-10	(Read) Do not access <sup>2</sup> (Write) UART Command Register	UCR <sub><i>n</i></sub>			
0x0C	27-12 27-12	(UART/Read) UART receive buffer (UART/Write) UART transmit buffer	URB <sub><i>n</i></sub> UTB <sub><i>n</i></sub>			
0x10	27-13 27-13	(Read) UART Input Port change Register (Write) UART Auxiliary Control Register <sup>1</sup>	UIPCR <sub><i>n</i></sub> UACR <sub><i>n</i></sub>			
0x14	27-14 27-14	(Read) UART Interrupt Status Register (Write) UART Interrupt Mask Register	UISR <sub><i>n</i></sub> UIMR <sub><i>n</i></sub>			
0x18	27-15	(Read) Do not access <sup>2</sup> UART Divider Upper Register	UBG1 <sub><i>n</i></sub>			
0x1C	27-15	(Read) Do not access <sup>2</sup> UART Divider Lower Register	UBG2 <sub><i>n</i></sub>			
0x34	27-16	(Read) UART Input Port Register (Write) Do not access <sup>2</sup>	UIP <sub><i>n</i></sub>			



**Table A-7. UART Memory Map (Continued)****Base Address:****UART0:** IPSBAR + 0x00\_0200**UART1:** IPSBAR + 0x00\_0240**UART2:** IPSBAR + 0x00\_0280

Offset	Page	Register Description	[31:24]	[23:16]	[15:8]	[7:0]
0x38	27-16	(Read) Do not access <sup>2</sup> (Write) UART Output Port Bit Set Command Register	UOP1n			
0x3C	27-16	(Read) Do not access <sup>2</sup> (Write) UART Output Port Bit Reset Command Register	UOP0n			

<sup>1</sup> UMR1n, UMR2n, and UCSRn should be changed only after the receiver/transmitter is issued a software reset command. That is, if channel operation is not disabled, undesirable results may occur.

<sup>2</sup> This address is for factory testing. Reading this location results in undesired effects and possible incorrect transmission or reception of characters. Register contents may also be changed.

**Table A-8. I<sup>2</sup>C Memory Map****Base Address: IPSBAR**

Offset	Page	Register Description	[31:24]	[23:16]	[15:8]	[7:0]
0x00_0300	26-8	I <sup>2</sup> C Address Register	I2ADR			
0x00_0304	26-9	I <sup>2</sup> C Frequency Divider Register	I2FDR			
0x00_0308	26-10	I <sup>2</sup> C Control Register	I2CR			
0x00_030C	26-11	I <sup>2</sup> C Status Register	I2SR			
0x00_0310	26-12	I <sup>2</sup> C Data I/O Register	I2DR			

**Table A-9. QSPI Memory Map****Base Address: IPSBAR**

Offset	Page	Register Description	[31:24]	[23:16]	[15:8]	[7:0]
0x00_0340	24-9	QSPI Mode Register	QMR			
0x00_0344	24-11	QSPI Delay Register	QDLYR			
0x00_0348	24-12	QSPI Wrap Register	QWR			
0x00_034C	24-12	QSPI Interrupt Register	QIR			

**Table A-9. QSPI Memory Map (Continued)****Base Address: IPSBAR**

Offset	Page	Register Description	[31:24]	[23:16]	[15:8]	[7:0]
0x00_0350	24-14	QSPI Address Register	QAR			
0x00_0354	24-14	QSPI Data Register	QDR			

**Table A-10. DMA Timer Memory Map****Base Address:****DMA Timer 0: IPSBAR + 0x00\_0400****DMA Timer 1: IPSBAR + 0x00\_0440****DMA Timer 2: IPSBAR + 0x00\_0480****DMA Timer 3: IPSBAR + 0x00\_04C0**

Offset	Page	Register Description	[31:24]	[23:16]	[15:8]	[7:0]
0x00	25-4 25-5 25-6	DMA Timer <i>n</i> Mode Register DMA Timer <i>n</i> Extended Mode Register DMA Timer <i>n</i> Event Register	DTMR <sub><i>n</i></sub>		DTXMR <sub><i>n</i></sub>	DTER <sub><i>n</i></sub>
0x04	25-7	DMA Timer <i>n</i> Reference Register	DTRR <sub><i>n</i></sub>			
0x08	25-8	DMA Timer <i>n</i> Capture Register	DTCR <sub><i>n</i></sub>			
0x0C	25-8	DMA Timer <i>n</i> Counter Register	DTCN <sub><i>n</i></sub>			

**Table A-11. INTC<sub>*n*</sub> Memory Map****Base Address:****INTC0: IPSBAR + 0x00\_0C00****INTC1: IPSBAR + 0x00\_0D00**

Offset	Page	Register Description	[31:24]	[23:16]	[15:8]	[7:0]
0x00	13-6	Interrupt Pending Register High	IPRH			
0x04	13-6	Interrupt Pending Register Low	IPRL			
0x08	13-7	Interrupt Mask Register High	IMRH			
0x0C	13-7	Interrupt Mask Register Low	IMRL			
0x10	13-9	Interrupt Force Register High	INTFRCH			
0x14	13-9	Interrupt Force Register Low	INTFRCL			

Table A-11. INTC<sub>n</sub> Memory Map (Continued)

Base Address:

INTC0: IPSBAR + 0x00\_0C00

INTC1: IPSBAR + 0x00\_0D00

Offset	Page	Register Description	[31:24]	[23:16]	[15:8]	[7:0]
0x18	13-11 13-11	Interrupt Request Level Register (IRLR <sub>n</sub> ) Interrupt Acknowledge Level and Priority Register (IACKLPR <sub>n</sub> )	IRLR[7:1]	IACKLPR[7:0]		
0x1C–0x3C	—					
0x40	13-12	Interrupt Control Registers		ICR01	ICR02	ICR03
0x44	13-12	Interrupt Control Registers	ICR04	ICR05	ICR06	ICR07
0x48	13-12	Interrupt Control Registers	ICR08	ICR09	ICR10	ICR11
0x4C	13-12	Interrupt Control Registers	ICR12	ICR13	ICR14	ICR15
0x50	13-12	Interrupt Control Registers	ICR16	ICR17	ICR18	ICR19
0x54	13-12	Interrupt Control Registers	ICR20	ICR21	ICR22	ICR23
0x58	13-12	Interrupt Control Registers	ICR24	ICR25	ICR26	ICR27
0x5C	13-12	Interrupt Control Registers	ICR28	ICR29	ICR30	ICR31
0x60	13-12	Interrupt Control Registers	ICR32	ICR33	ICR34	ICR35
0x64	13-12	Interrupt Control Registers	ICR36	ICR37	ICR38	ICR39
0x68	13-12	Interrupt Control Registers	ICR40	ICR41	ICR42	ICR43
0x6C	13-12	Interrupt Control Registers	ICR44	ICR45	ICR46	ICR47
0x70	13-12	Interrupt Control Registers	ICR48	ICR49	ICR50	ICR51
0x74	13-12	Interrupt Control Registers	ICR52	ICR53	ICR54	ICR55
0x78	13-12	Interrupt Control Registers	ICR56	ICR57	ICR58	ICR59
0x7C	13-12	Interrupt Control Registers	ICR60	ICR61	ICR62	ICR63
0x80 - 0xDC						
0xE0	13-15	Software IACK Register	SWIACK			
0xE4	13-15	Level 1 IACK Register	L1IACK			
0xE8	13-15	Level 2 IACK Register	L2IACK			

**Table A-11. INTC<sub>n</sub> Memory Map (Continued)****Base Address:****INTC0: IPSBAR + 0x00\_0C00****INTC1: IPSBAR + 0x00\_0D00**

Offset	Page	Register Description	[31:24]	[23:16]	[15:8]	[7:0]
0xEC	13-15	Level 3 IACK Register	L3IACK			
0xF0	13-15	Level 4 IACK Register	L4IACK			
0xF4	13-15	Level 5 IACK Register	L5IACK			
0xF8	13-15	Level 6 IACK Register	L6IACK			
0xFC	13-15	Level 7 IACK Register	L7IACK			

**Table A-12. FEC Memory Map****Base Address:****FEC0: IPSBAR + 0x00\_1000****FEC1: IPSBAR + 0x00\_1800**

Offset	Page	Register Description	[31:24]	[23:16]	[15:8]	[7:0]
0x004	19-9	Interrupt Event Register	EIR <sub>n</sub>			
0x008	19-10	Interrupt Mask Register	EIMR <sub>n</sub>			
0x010	19-11	Receive Descriptor Active Register	RDAR <sub>n</sub>			
0x014	19-12	Transmit Descriptor Active Register	TDAR <sub>n</sub>			
0x024	19-13	Ethernet Control Register	ECR <sub>n</sub>			
0x040	19-14	MII Data Register	MDATA <sub>n</sub>			
0x044	19-16	MII Speed Control Register	MSCR <sub>n</sub>			
0x064	19-17	MIB Control/Status Register	MIBC <sub>n</sub>			
0x084	19-18	Receive Control Register	RCR <sub>n</sub>			
0x0C4	19-19	Transmit Control Register	TCR <sub>n</sub>			
0x0E4	19-20	Physical Address Low Register	PALR <sub>n</sub>			
0x0E8	19-21	Physical Address High+ Type Field	PAUR <sub>n</sub>			
0x0EC	19-21	Opcode + Pause Duration	OPD <sub>n</sub>			
0x118	19-22	Upper 32 bits of Individual Hash Table	IAUR <sub>n</sub>			

Table A-12. FEC Memory Map (Continued)

Base Address:

FEC0: IPSBAR + 0x00\_1000

FEC1: IPSBAR + 0x00\_1800

Offset	Page	Register Description	[31:24]	[23:16]	[15:8]	[7:0]
0x11C	19-23	Lower 32 Bits of Individual Hash Table	IALR <sub>n</sub>			
0x120	19-23	Upper 32 bits of Group Hash Table	GAUR <sub>n</sub>			
0x124	19-24	Lower 32 bits of Group Hash Table	GALR <sub>n</sub>			
0x144	19-25	Transmit FIFO Watermark	TFWR <sub>n</sub>			
0x14C	19-25	FIFO Receive Bound Register	FRBR <sub>n</sub>			
0x150	19-26	FIFO Receive FIFO Start Registers	FRSR <sub>n</sub>			
0x180	19-27	Pointer to Receive Descriptor Ring	ERDSR <sub>n</sub>			
0x184	19-27	Pointer to Transmit Descriptor Ring	ETDSR <sub>n</sub>			
0x188	19-28	Maximum Receive Buffer Size	EMRBR <sub>n</sub>			

Table A-13. FEC MIB Counters Memory Map

Base Address:

FEC0: IPSBAR + 0x00\_1200

FEC1: IPSBAR + 0x00\_1A00

Offset	Page	Register Description	[31:24]	[23:16]	[15:8]	[7:0]
0x00	19-7	Count of frames not counted correctly	RMON_T_DROP <sub>n</sub>			
0x04	19-7	RMON Tx packet count	RMON_T_PACKETS <sub>n</sub>			
0x08	19-7	RMON Tx Broadcast Packets	RMON_T_BC_PKT <sub>n</sub>			
0x0C	19-7	RMON Tx Multicast Packets	RMON_T_MC_PKT <sub>n</sub>			
0x10	19-7	RMON Tx Packets w CRC/Align error	RMON_T_CRC_ALIGN <sub>n</sub>			
0x14	19-7	RMON Tx Packets < 64 bytes, good crc	RMON_T_UNDERSIZE <sub>n</sub>			
0x18	19-7	RMON Tx Packets > MAX_FL bytes, good crc	RMON_T_OVERSIZE <sub>n</sub>			
0x1C	19-7	RMON Tx Packets < 64 bytes, bad crc	RMON_T_FRAG <sub>n</sub>			
0x20	19-7	RMON Tx Packets > MAX_FL bytes, bad crc	RMON_T_JAB <sub>n</sub>			
0x24	19-7	RMON Tx collision count	RMON_T_COL <sub>n</sub>			

Table A-13. FEC MIB Counters Memory Map (Continued)

Base Address:

FEC0: IPSBAR + 0x00\_1200

FEC1: IPSBAR + 0x00\_1A00

Offset	Page	Register Description	[31:24]	[23:16]	[15:8]	[7:0]
0x28	19-7	RMON Tx 64 byte packets	RMON_T_P64n			
0x2C	19-7	RMON Tx 65 to 127 byte packets	RMON_T_P65TO127n			
0x30	19-7	RMON Tx 128 to 255 byte packets	RMON_T_P128TO255n			
0x34	19-7	RMON Tx 256 to 511 byte packets	RMON_T_P256TO511n			
0x38	19-7	RMON Tx 512 to 1023 byte packets	RMON_T_P512TO1023n			
0x3C	19-7	RMON Tx 1024 to 2047 byte packets	RMON_T_P1024TO2047n			
0x40	19-7	RMON Tx packets w > 2048 bytes	RMON_T_P_GTE2048n			
0x44	19-7	RMON Tx Octets	RMON_T_OCTETSn			
0x48	19-7	Count of frames not counted correctly	IEEE_T_DROPn			
0x4C	19-7	Frames Transmitted OK	IEEE_T_FRAME_OKn			
0x50	19-7	Frames Transmitted with Single Collision	IEEE_T_1COLn			
0x54	19-7	Frames Transmitted with Multiple Collisions	IEEE_T_MCOLn			
0x58	19-7	Frames Transmitted after Deferral Delay	IEEE_T_DEFn			
0x5C	19-7	Frames Transmitted with Late Collision	IEEE_T_LCOLn			
0x60	19-7	Frames Transmitted with Excessive Collisions	IEEE_T_EXCOLn			
0x64	19-7	Frames Transmitted with Tx FIFO Underrun	IEEE_T_MACERRn			
0x68	19-7	Frames Transmitted with Carrier Sense Error	IEEE_T_CSERRn			
0x6C	19-7	Frames Transmitted with SQE Error	IEEE_T_SQEn			
0x70	19-7	Flow Control Pause frames transmitted	IEEE_T_FDXFCn			
0x74	19-7	Octet count for Frames Transmitted w/o Error	IEEE_T_OCTETS_OKn			
0x78– 0x80	19-7					
0x84	19-7	RMON Rx packet count	RMON_R_PACKETSn			
0x88	19-7	RMON Rx Broadcast Packets	RMON_R_BC_PKTn			

Table A-13. FEC MIB Counters Memory Map (Continued)

Base Address:

FEC0: IPSBAR + 0x00\_1200

FEC1: IPSBAR + 0x00\_1A00

Offset	Page	Register Description	[31:24]	[23:16]	[15:8]	[7:0]
0x8C	19-7	RMON Rx Multicast Packets	RMON_R_MC_PKT $n$			
0x90	19-7	RMON Rx Packets w CRC/Align error	RMON_R_CRC_ALIGN $n$			
0x94	19-7	RMON Rx Packets < 64 bytes, good crc	RMON_R_UNDERSIZE $n$			
0x98	19-7	RMON Rx Packets > MAX_FL bytes, good crc	RMON_R_OVERSIZE $n$			
0x9C	19-7	RMON Rx Packets < 64 bytes, bad crc	RMON_R_FRAG $n$			
0xA0	19-7	RMON Rx Packets > MAX_FL bytes, bad crc	RMON_R_JAB $n$			
0xA4	19-7		RMON_R_RESVD_0 $n$			
0xA8	19-7	RMON Rx 64 byte packets	RMON_R_P64 $n$			
0xAC	19-7	RMON Rx 65 to 127 byte packets	RMON_R_P65TO127 $n$			
0xB0	19-7	RMON Rx 128 to 255 byte packets	RMON_R_P128TO255 $n$			
0xB4	19-7	RMON Rx 256 to 511 byte packets	RMON_R_P256TO511 $n$			
0xB8	19-7	RMON Rx 512 to 1023 byte packets	RMON_R_P512TO1023 $n$			
0xBC	19-7	RMON Rx 1024 to 2047 byte packets	RMON_R_P1024TO2047 $n$			
0xC0	19-7	RMON Rx packets w > 2048 bytes	RMON_R_P_GTE2048 $n$			
0xC4	19-7	RMON Rx Octets	RMON_R_OCTETS $n$			
0xC8	19-7	Count of frames not counted correctly	IEEE_R_DROP $n$			
0xCC	19-7	Frames Received OK	IEEE_R_FRAME_OK $n$			
0xD0	19-7	Frames Received with CRC Error	IEEE_R_CRC $n$			
0xD4	19-7	Frames Received with Alignment Error	IEEE_R_ALIGN $n$			
0xD8	19-7	Receive Fifo Overflow count	IEEE_R_MACERR $n$			
0xDC	19-7	Flow Control Pause frames received	IEEE_R_FDXFC $n$			
0xE0	19-7	Octet count for Frames Rcvd w/o Error	IEEE_R_OCTETS_OK $n$			

Table A-14. GPIO (Ports) Memory Map

Base Address: IPSBAR

Offset	Page	Register Description	[31:24]	[23:16]	[15:8]	[7:0]
0x0000	12-11	Port Output Data Registers				
0x0004			PODR_BUSCTL	PODR_ADDR		
0x0008			PODR_CS		PODR_FEC0H	PODR_FEC0L
0x000C			PODR_FECI2C	PODR_QSPI	PODR_SDRAM	PODR_TIMERH
0x0010			PODR_TIMERL	PODR_UARTL	PODR_FEC1H	PODR_FEC1L
0x0014			PODR_BS		PODR_USBH	PODR_USBL
0x0018			PODR_earth			
0x001C	12-14	Port Data Direction Registers				
0x0020			PDDR_BUSCTL	PDDR_ADDR		
0x0024			PDDR_CS		PDDR_FEC0H	PDDR_FEC0L
0x0028			PDDR_FECI2C	PDDR_QSPI	PDDR_SDRAM	PDDR_TIMERH
0x002C			PDDR_TIMERL	PDDR_UARTL	PDDR_FEC1H	PDDR_FEC1L
0x0030			PDDR_BS		PDDR_USBH	PDDR_USBL
0x0034			PDDR_earth			
0x0038	12-16	Port Pin Data/Set Data Registers				
0x003C			PPDSDR_BUSCTL	PPDSDR_ADDR		
0x0040			PPDSDR_CS		PPDSDR_FEC0H	PPDSDR_FEC0L
0x0044			PPDSDR_FECI2C	PPDSDR_QSPI	PPDSDR_SDRA M	PPDSDR_TIMERH
0x0048			PPDSDR_TIMERL	PPDSDR_UARTL	PPDSDR_FEC1H	PPDSDR_FEC1L
0x004C			PPDSDR_BS		PPDSDR_USBH	PPDSDR_USBL
0x0050			PPDSDR_earth			



Table A-14. GPIO (Ports) Memory Map (Continued)

Base Address: IPSBAR

Offset	Page	Register Description	[31:24]	[23:16]	[15:8]	[7:0]
0x0054	12-19	Port Clear Output Data Registers				
0x0058			PPDSDR_BUSCTL	PPDSDR_ADDR		
0x005C			PPDSDR_CS		PPDSDR_FEC0H	PPDSDR_FEC0L
0x0060			PPDSDR_FECI2C	PPDSDR_QSPI	PPDSDR_SDRAM	PPDSDR_TIMERH
0x0064			PPDSDR_TIMERL	PPDSDR_UARTL	PPDSDR_FEC1H	PPDSDR_FEC1L
0x0068			PPDSDR_BS		PPDSDR_USBH	PPDSDR_USBL
0x006C			PPDSDR_earth			
0x0070	12-21	Pin Assignment Registers	PAR_ADDR	PAR_CS	PAR_BUSCTL	
0x0074						PAR_USB
0x0078			PAR_FEC0HL	PAR_FEC1HL	PAR_TIMER	
0x007C			PAR_UART			PAR_QSPI
0x0080			PAR_SDRAM			PAR_FECI2C
0x0084			PAR_BS			

Table A-15. CCM Memory Map

Base Address: IPSBAR

Offset	Page	Register Description	[31:24]	[23:16]	[15:8]	[7:0]
0x11_0004	9-4 8-3	Chip Configuration Register Low-Power Control Register	CCR		LPCR <sup>1</sup>	
0x11_0008	9-5 9-7	Reset Configuration Register Chip Identification Register	RCON		CIR	
0x11_000C	—		Reserved <sup>2</sup>			
0x11_0010	—		Unimplemented <sup>3</sup>			

<sup>1</sup> See Chapter 8, "Power Management," for a description of the LPCR. It is shown here only to warn against accidental writes to this register.<sup>2</sup> Writing to reserved addresses with values other than 0 could put the device in a test mode; reading returns 0s.<sup>3</sup> Accessing an unimplemented address has no effect and causes a cycle termination transfer error.

**Table A-16. Clocks Memory Map****Base Address: IPSBAR**

Offset	Page	Register Description	[31:24]	[23:16]	[15:8]	[7:0]
0x12_0000	7-8	Synthesizer Control Register	SYNCR			
0x12_0004	7-11	Synthesizer Status Register	SYNSR			

**Table A-17. Edge Port Memory Map****Base Address: IPSBAR**

Offset	Page	Register Description	[31:24]	[23:16]	[15:8]	[7:0]
0x13_0000	14-3	EPORT Pin Assignment Register	EPPAR		EPDDR	EPIER
	14-4	EPORT Data Direction Register				
	14-5	EPORT Interrupt Enable Register				
0x13_0004	14-5	EPORT Data Register	EPDR	EPPDR	EPFR	
	14-6	EPORT Pin Data Register				
	14-6	EPORT Flag Register				

**Table A-18. Watchdog Timer Memory Map****Base Address: IPSBAR**

Offset	Page	Register Description	[31:24]	[23:16]	[15:8]	[7:0]
0x14_0000	21-3	Watchdog Control Register	WCR		WMR	
	21-4	Watchdog Modulus Register				
0x14_0004	21-4	Watchdog Count Register	WCNTR		WSR	
	21-4	Watchdog Service Register				

**Table A-19. PIT<sub>n</sub> Memory Map****Base Address:****PIT0:** IPSBAR + 0x15\_0000**PIT1:** IPSBAR + 0x16\_0000**PIT2:** IPSBAR + 0x17\_0000**PIT3:** IPSBAR + 0x18\_0000

Offset	Page	Register Description	[31:24]	[23:16]	[15:8]	[7:0]
0x0000	23-3 23-5	PIT Control and Status Register PIT Modulus Register	PCSR <sub>n</sub>		PMR <sub>n</sub>	
0x0004	23-5	PIT Count Register	PCNTR <sub>n</sub>			

**Table A-20. MDHA Memory Map****Base Address: IPSBAR**

Offset	Page	Register Description	[31:24]	[23:16]	[15:8]	[7:0]
0x19_0000	28-4	MDHA Mode Register	MDMR			
0x19_0004	28-6	MDHA Control Register	MDCR			
0x19_0008	28-7	MDHA Command Register	MDCMR			
0x19_000C	28-8	MDHA Status Register	MDSR			
0x19_0010	28-10	MDHA Interrupt Status Register	MDISR			
0x19_0014	28-10	MDHA Interrupt Mask Register	MDIMR			
0x19_001C	28-11	MDHA Data Size Register	MDDSR			
0x19_0020– 0x19_002C	28-12	MDHA Input FIFO Register	MDIN			
0x19_0030	28-12	Message Digest A0 Register	MDA0			
0x19_0034	28-12	Message Digest B0 Register	MDB0			
0x19_0038	28-12	Message Digest C0 Register	MDC0			
0x19_003C	28-12	Message Digest D0 Register	MDD0			
0x19_0040	28-12	Message Digest E0 Register	MDE0			
0x19_0044	28-12	Message Data Size Register	MDMDS			
0x19_0048– 0x19_006C	—					

**Table A-20. MDHA Memory Map (Continued)****Base Address: IPSBAR**

Offset	Page	Register Description	[31:24]	[23:16]	[15:8]	[7:0]
0x19_0070	<a href="#">28-13</a>	Message Digest A1 Register	MDA1			
0x19_0074	<a href="#">28-13</a>	Message Digest B1 Register	MDB1			
0x19_0078	<a href="#">28-13</a>	Message Digest C1 Register	MDC1			
0x19_007C	<a href="#">28-13</a>	Message Digest D1 Register	MDD1			
0x19_0080	<a href="#">28-13</a>	Message Digest E1 Register	MDE1			

**Table A-21. RNG Memory Map****Base Address: IPSBAR**

Offset	Page	Register Description	[31:24]	[23:16]	[15:8]	[7:0]
0x1A_0000	<a href="#">30-1</a>	RNG Control Register	RNGCR			
0x1A_0004	<a href="#">30-2</a>	RNG Status Register	RNGSR			
0x1A_0008	<a href="#">30-3</a>	RNG Entropy Register	RNGER			
0x1A_000C	<a href="#">30-4</a>	RNG Output FIFO	RNGOUT			

**Table A-22. SKHA Memory Map****Base Address: IPSBAR**

Offset	Page	Register Description	[31:24]	[23:16]	[15:8]	[7:0]
0x1B_0000	<a href="#">29-7</a>	SKHA Mode Register	SKMR			
0x1B_0004	<a href="#">29-8</a>	SKHA Control Register	SKCR			
0x1B_0008	<a href="#">29-9</a>	SKHA Command Register	SKCMR			
0x1B_000C	<a href="#">29-10</a>	SKHA Status Register	SKSR			
0x1B_0010	<a href="#">29-11</a>	SKHA Interrupt Status Register	SKESR			
0x1B_0014	<a href="#">29-12</a>	SKHA Interrupt Mask Register	SKEMR			
0x1B_0018	<a href="#">29-13</a>	SKHA Key Size Register	SKKSR			
0x1B_001C	<a href="#">29-13</a>	SKHA Data Size Register	SKDSR			
0x1B_0020	<a href="#">29-14</a>	SKHA Input FIFO	SKIN			

Table A-22. SKHA Memory Map (Continued)

Base Address: IPSBAR

Offset	Page	Register Description	[31:24]	[23:16]	[15:8]	[7:0]
0x1B_0024	<a href="#">29-14</a>	SKHA Output FIFO	SKOUT			
0x1B_0030	<a href="#">29-14</a>	SKHA Key 1	SKK1			
0x1B_0034	<a href="#">29-14</a>	SKHA Key 2	SKK2			
0x1B_0038	<a href="#">29-14</a>	SKHA Key 3	SKK3			
0x1B_003C	<a href="#">29-14</a>	SKHA Key 4	SKK4			
0x1B_0040	<a href="#">29-14</a>	SKHA Key 5	SKK5			
0x1B_0044	<a href="#">29-14</a>	SKHA Key 6	SKK6			
0x1B_0048– 0x1B_006C	—					
0x1B_0070	<a href="#">29-15</a>	SKHA Context 1 (IV/Nonce/Running Offset/CTR/MAC)	SKC1			
0x1B_0074	<a href="#">29-15</a>	SKHA Context 2	SKC2			
0x1B_0078	<a href="#">29-15</a>	SKHA Context 3	SKC3			
0x1B_007C	<a href="#">29-15</a>	SKHA Context 4	SKC4			
0x1B_0080	<a href="#">29-15</a>	SKHA Context 5	SKC5			
0x1B_0084	<a href="#">29-15</a>	SKHA Context 6	SKC6			
0x1B_0088	<a href="#">29-15</a>	SKHA Context 7	SKC7			
0x1B_008C	<a href="#">29-15</a>	SKHA Context 8	SKC8			
0x1B_0090	<a href="#">29-15</a>	SKHA Context 9	SKC9			
0x1B_0094	<a href="#">29-15</a>	SKHA Context 10	SKC10			
0x1B_0098	<a href="#">29-15</a>	SKHA Context 11	SKC11			
0x1B_009C	<a href="#">29-15</a>	SKHA Context 12	SKC12			

Table A-23. USB Memory Map

Base Address: IPSBAR

Offset	Page	Register Description	[31:24]	[23:16]	[15:8]	[7:0]
0x1C_0000	20-6	USB Frame Number and Match	USB_FRAME			
0x1C_0004	20-6	USB Specification/Release Number	USB_SPEC			
0x1C_0008	20-7	USB Status Register	USB_STAT			
0x1C_000C	20-7	USB Control Registers	USB_CTRL			
0x1C_0010	20-9	USB Descriptor RAM Address	USB_DADR			
0x1C_0014	20-10	USB Descriptor RAM/Endpoint Buffer Data	USB_DDAT			
0x1C_0018	20-10	USB Interrupt Status Register	USB_INTR			
0x1C_001C	20-12	USB Interrupt Mask Register	USB_MASK			
0x1C_0020	20-13	USB FIFO Memory Control	USB_MCTL			
0x1C_0024– 0x1C_002C	—					
0x1C_0030+(n*0x30) <sup>1</sup>	20-13	Endpoint <i>n</i> Status/Control	USB_EP <sub><i>n</i></sub> _STAT			
0x1C_0034+(n*0x30)	20-15	Endpoint <i>n</i> Interrupt Status	USB_EP <sub><i>n</i></sub> _INTR			
0x1C_0038+(n*0x30)	20-17	Endpoint <i>n</i> Interrupt Mask	USB_EP <sub><i>n</i></sub> _MASK			
0x1C_003C+(n*0x30)	20-17	Endpoint <i>n</i> FIFO Data	USB_EP <sub><i>n</i></sub> _FDAT			
0x1C_0040+(n*0x30)	20-18	Endpoint <i>n</i> FIFO Status	USB_EP <sub><i>n</i></sub> _FSTAT			
0x1C_0044+(n*0x30)	20-19	Endpoint <i>n</i> FIFO Control	USB_EP <sub><i>n</i></sub> _FCTRL			
0x1C_0048+(n*0x30)	20-21	Endpoint <i>n</i> FIFO Last Read Frame Pointer	USB_EP <sub><i>n</i></sub> _LRFP			
0x1C_004C+(n*0x30)	20-21	Endpoint <i>n</i> FIFO Last Write Frame Pointer	USB_EP <sub><i>n</i></sub> _LWFP			
0x1C_0050+(n*0x30)	20-22	Endpoint <i>n</i> FIFO Alarm	USB_EP <sub><i>n</i></sub> _FALRM			
0x1C_0054+(n*0x30)	20-23	Endpoint <i>n</i> FIFO Read Pointer	USB_EP <sub><i>n</i></sub> _FRDP			
0x1C_0058+(n*0x30)	20-24	Endpoint <i>n</i> FIFO Write Pointer	USB_EP <sub><i>n</i></sub> _FWRP			
0x1C_005C+(n*0x30)	—					

<sup>1</sup> 'n' refers to the number of endpoints, 0, 1, 2, and 3.

**Table A-24. PWM Memory Map****Base Address: IPSBAR**

Offset	Page	Register Description	[31:24]	[23:16]	[15:8]	[7:0]
0x1D_0000	22-2 22-3 22-4 22-4	PWM Enable Register PWM Polarity Register PWM Clock Select Register PWM Prescale Clock Select Register	PWME	PWMPOL	PWMCLK	PWMPRCLK
0x1D_0004	22-5 22-6	PWM Center Align Enable Register PWM Control Register	PWMCAE	PWMCTL		
0x1D_0008	22-7 22-7	PWM Scale A Register PWM Scale B Register	PWMSCLA	PWMSCLB		
0x1D_000C	22-8	PWM Channel 0 Counter Register PWM Channel 1 Counter Register PWM Channel 2 Counter Register PWM Channel 3 Counter Register	PWMCNT0	PWMCNT1	PWMCNT2	PWMCNT3
0x1D_0010	22-9	PWM Channel 0 Period Register PWM Channel 1 Period Register			PWMPER0	PWMPER1
0x1D_0014	22-9	PWM Channel 2 Period Register PWM Channel 3 Period Register	PWMPER2	PWMPER3		
0x1D_0018	22-10	PWM Channel 0 Duty Register PWM Channel 1 Duty Register PWM Channel 2 Duty Register PWM Channel 3 Duty Register	PWMDTY0	PWMDTY1	PWMDTY2	PWMDTY3





# Index

## B

BDM, *see* Debug

### Bus

- arbitration 11-9–11-12, ??–11-13
  - fixed mode 11-11
  - round-robin mode 11-10
- characteristics 16-2
- data transfer
  - back-to-back cycles 16-10
  - burst cycles 16-11
    - line read bus cycles 16-12
    - line transfers 16-12
    - line write bus cycles 16-14
  - cycle execution 16-4
  - cycle states 16-5
  - fast termination cycle 16-9
  - read cycle 16-7
  - write cycle 16-8
- operands, misaligned 16-17
- SACU 11-13

## C

### Cache

- block diagram 5-3
- coherency 5-4
- fill buffer 5-2, 5-5
- invalidation 5-4
- miss fetch algorithm 5-5
- organization 5-1
- registers
  - access control 0–1 (ACR<sub>n</sub>) 3-8, 5-10
  - control (CACR) 3-7, 5-7
- SRAM interaction 5-3

### CCM

- configuration
  - boot device 9-10
  - chip mode 9-9
  - chip select 9-10
  - clock mode 9-10
  - output pad strength 9-10
  - reset 9-7
- memory map 9-3
- operation
  - low-power modes 8-9
- registers
  - chip configuration (CCR) 9-4

- chip identification (CIR) 9-7
- reset configuration (RCON) 9-5

### signals

- CLKMOD1–0 9-2
- RCON 9-2

### Chip select module

- byte strobes ( $\overline{BS}3-0$ ) 15-2
- code example 15-11
- memory map 15-6
- operation
  - general 15-3
  - low-power modes 8-7
  - port sizing 15-4
  - wait state 15-4
- registers
  - address (CSAR<sub>n</sub>) 15-7
  - control (CSCR<sub>n</sub>) 15-9
  - mask (CSMR<sub>n</sub>) 15-8
- signals
  - chip select ( $\overline{CS}7-0$ ) 15-1
  - output enable ( $\overline{OE}$ ) 15-1

### Clock module

- memory map 7-8
- operation
  - low-power modes 7-6, 8-10
  - normal PLL mode 7-5
  - reset 7-14

### PLL

- charge pump/loop filter 7-22, 7-23
- lock detection 7-23
- loss-of-clock
  - alternate clock selection 7-25
  - detection 7-25
  - reset 7-25
  - stop mode 7-26
- loss-of-lock
  - conditions 7-24
  - reset 7-25
- multiplication factor divider (MFD) 7-23
- operation 7-21
- phase and frequency detector (PFD) 7-22
- voltage control output (VCO) 7-23
- registers
  - synthesizer control (SYNCR) 7-8
  - synthesizer status (SYNSR) 7-11
- signals
  - CLKMOD1–0 7-7

- CLKOUT 7-7
- EXTAL 7-7
- RSTOUT 7-7
- XTAL 7-7
- system clock 7-15
- Collision handling 19-47
- Core
  - block diagram 3-1
  - low-power modes 8-6
  - pipelines 3-1
  - registers
    - address ( $A_n$ ) 3-3
    - condition code (CCR) 3-4
    - data ( $D_n$ ) 3-2
    - program counter (PC) 3-3
    - stack pointer (A7) 3-3, 3-7
    - status register (SR) 3-6
    - vector base (VBR) 3-7

**D**

- Debug
  - BDM
    - commands
      - DUMP 31-27
      - FILL 31-29
      - format 31-21
      - GO 31-31
      - NOP 31-31
      - RAREG/RDREG 31-23
      - RCREG 31-32
      - RDMREG 31-35
      - READ 31-25
      - summary 31-19
      - WAREG/WDREG 31-24
      - WCREG 31-34
      - WDMREG 31-36
      - WRITE 31-26
    - CPU halt 31-16
    - receive packet 31-18
    - recommended pinout 31-45
    - serial interface 31-17
    - transmit packet 31-19
  - low-power modes 8-11
  - memory map 31-5
  - real-time support 31-37–31-40
  - registers
    - address attribute trigger (AATR) 31-7
    - address breakpoint (ABLR, ABHR) 31-8
    - configuration/status (CSR) 31-9
    - data breakpoint/mask (DBR, DBMR) 31-12
    - program counter breakpoint/mask (PBR, PBMR) 31-13
    - trigger definition (TDR) 31-14
  - signals

- breakpoint ( $\overline{BKPT}$ ) 31-2
- debug data (DDATA3–0) 31-2
- development serial clock (DSCLK) 31-2
- development serial input (DSI) 31-2
- development serial output (DSO) 31-2
- processor status (PST3–0) 31-2
- PSTCLK 31-2
- taken branch 31-4
- trace 31-2
- DMA
  - data transfer
    - requests 18-16
- DMA controller
  - channel prioritization 18-15
  - data transfer 18-4
    - auto alignment 18-19
    - bandwidth control 18-20
    - requests 18-14
    - termination 18-20
  - low-power modes 8-7
  - programming 18-15
  - registers
    - byte count (BCR $_n$ ) 18-8
    - control (DCR $_n$ ) 18-10
    - destination address (DAR $_n$ ) 18-8
    - request control (DMAREQC) 18-6
    - source address (SAR $_n$ ) 18-7
    - status (DSR $_n$ ) 18-9

**E**

- EMAC
  - data representation 4-14
  - instructions
    - execution timing 3-25, 4-13
    - summary 4-12
  - MAC, comparison 4-1
  - memory map 4-6
  - opcodes 4-14
  - operation
    - fractional 4-9
    - general 4-3
  - registers
    - mask (MASK) 4-11
    - status (MACSR) 4-6
- EPORT
  - low-power modes 8-10, 14-1
  - memory map 14-2
  - registers
    - data direction (EPDDR) 14-4
    - flag (EPFR) 14-6
    - pin assignment (EPPAR) 14-3
    - pin data (EPPDR) 14-6
    - port data (EPDR) 14-5
    - port interrupt enable (EPIER) 14-5

- Ethernet
    - address recognition 19-40
    - block diagram 19-2
    - buffer descriptors
      - receive (RxBD) 19-31
      - transmit (TxBD) 19-33
    - collision handling 19-47
    - errors
      - handling 19-48
      - reception
        - CRC 19-49
        - frame length 19-49
        - non-octet 19-49
        - overrun 19-49
        - truncation 19-50
      - transmission
        - attempts limit expired 19-48
        - heartbeat 19-49
        - late collision 19-48
        - underrun 19-48
    - frame reception 19-39
    - frame transmission 19-38
    - hash table 19-43
    - initialization 19-35
    - memory map 19-6
    - operation
      - 10 Mbps 7-Wire 19-5, 19-37
      - 10 Mbps and 100 Mbps MII 19-37
      - full duplex 19-4, 19-46
      - half duplex 19-4
      - loopback 19-47
      - low-power modes 8-9
    - registers
      - control (ECR) 19-13
      - descriptor group upper/lower address (GAUR/GALR) 19-24
      - descriptor individual upper/lower (IAUR/IALR) 19-23
      - descriptor individual upper/lower address (IAUR/IALR) 19-22
      - FIFO receive bound (FRBR) 19-25
      - FIFO receive start (FRSR) 19-26
      - FIFO transmit FIFO watermark (TFWR) 19-25
      - interrupt event (EIR) 19-9
      - interrupt mask (EIMR) 19-10
      - MIB control (MIBC) 19-17
      - II management frame (MMFR) 19-14
      - II speed control (MSCR) 19-16
      - opcode/pause duration (OPD) 19-21
      - physical address low (PALR<sub>n</sub>) 19-20
      - physical address low/high (PALR, PAUR) 19-20
      - receive buffer size (EMRBR) 19-28
      - receive control (RCR) 19-18
      - receive descriptor active (RDAR) 19-11
      - receive descriptor ring start (ERDSR) 19-27
      - transmit buffer descriptor ring start (ETSDR) 19-27
      - transmit control (TCR) 19-19
      - transmit descriptor active (TDAR) 19-12
  - Exceptions
    - access error 3-13
    - divide-by-zero 3-14
    - exception stack frame 3-11
    - format error 3-15
    - illegal instruction 3-13
    - overview 3-9
    - privilege violation 3-14
    - reset 3-16
    - trace 3-14
    - TRAP instruction 3-15
  - External interface module (EIM), *see* bus
- F**
- Fault-on-fault 3-16
  - FEC, *see* Ethernet
- G**
- GPIO
- block diagram 12-2
  - memory map 12-10
  - operation
    - low-power modes 8-9
  - registers
    - pin assignment (PAR<sub>x</sub>) 12-21
    - port clear output data (PCLRR<sub>x</sub>) 12-19
    - port data direction (PDDR<sub>x</sub>) 12-14
    - port output data (PODR<sub>x</sub>) 12-11
    - port pin data/set data (PPDSR<sub>x</sub>) 12-16
- I**
- I<sup>2</sup>C
- arbitration procedure 26-6
  - clock
    - arbitration 26-6
    - stretching 26-8
    - synchronization 26-6
  - handshaking 26-8
  - memory map 26-8
  - operation 26-3
    - low-power modes 8-7
  - programming examples
    - initialization 26-13
    - repeated START generation 26-16
    - START generation 26-14
    - STOP generation 26-15
  - registers
    - address (I2ADR) 26-8

- control (I2CR) 26-10
  - data I/O (I2DR) 26-12
  - frequency divider (I2FDR) 26-9
  - status (I2SR) 26-11
  - slave mode 26-16
  - Instructions
    - additions 3-9
    - enhancements
      - BITREV 3-27
      - BYTEREV 3-28
      - FF1 3-29
      - STRLDSR 3-30
    - execution timing
      - branch 3-26
      - EMAC 3-25
      - miscellaneous 3-24
      - MOVE 3-20
      - one-operand 3-21
      - two-operand 3-22
  - JTAG
    - BYPASS 32-10
    - CLAMP 32-10
    - ENABLE\_TEST\_CTRL 32-10
    - EXTTEST 32-9
    - HIGHZ 32-10
    - IDCODE 32-9
    - SAMPLE/PRELOAD 32-9
  - PULSE 31-3
  - RTE 3-15
  - Interrupt controller
    - interrupts
      - debug 3-15
      - PIT 23-7
      - prioritization 13-3
      - recognition 13-3
      - sources 13-13
      - vector determination 13-4
    - memory map 13-4
    - operation
      - low-power modes 8-8
    - overview 13-1–13-2
    - registers
      - (IACKLPR<sub>n</sub>) 13-11, 1-9
      - interrupt control (ICR<sub>nx</sub>) 13-12
      - interrupt force high/low (INTFRCH<sub>n</sub>, INTFRCL<sub>n</sub>) 13-9
      - interrupt pending high/low (IPRH<sub>n</sub>, IPRL<sub>n</sub>) 13-6
      - interrupt request level (IRLR<sub>n</sub>) 13-11
      - mask high/low (IMRH<sub>n</sub>, n) 13-7
- J**
- JTAG
    - instructions
      - BYPASS 32-10
      - CLAMP 32-10
      - ENABLE\_TEST\_CTRL 32-10
      - EXTTEST 32-9
      - HIGHZ 32-10
      - IDCODE 32-9
      - SAMPLE/PRELOAD 32-9
    - low-power modes 8-11
    - memory map 32-5
    - nonscan chain operation 32-11
    - registers
      - bypass 32-6
      - IDCODE 32-5
      - instruction shift (IR) 32-5
      - TEST\_CTRL 32-6
    - signals
      - JTAG\_EN 32-3
      - TCLK 32-4
      - test data input/development serial input (TDI/DSI) 32-4
      - test data output/development serial output (TDO/DSO) 32-5
      - test reset/development serial clock (TRST/DSCLK) 32-4
    - TAP controller 32-7
- L**
- Low-power modes
    - doze 8-5
    - peripheral behavior
      - CCM 8-9
      - chip select module 8-7
      - clock module 8-10
      - core 8-6
      - debug 8-11
      - DMA controller 8-7
      - DTIM 8-8
      - EPORT 8-10
      - Ethernet 8-9
      - GPIO 8-9
      - I<sup>2</sup>C 8-7
      - interrupt controller 8-8
      - JTAG 8-11
      - modules 8-7
      - PIT 8-10
      - QSPI 8-8
      - reset controller 8-9
      - SCM 8-6
      - SDRAM controller 8-6
      - SRAM 8-6
      - WDT 8-10
    - run 8-5
    - stop 8-5
    - summary 8-11
    - wait 8-5

## M

MAC, *see* EMAC

### MDHA

- logic 28-14
- memory map 28-3
- modes 28-2
- operation
  - HASH 28-15
  - HMAC 28-15
  - IPAD 28-16
  - MAC 28-18
  - NMAC 28-19
  - OPAD 28-16
  - SHA-1 EHMAC 28-17

### registers

- command (MDCMR) 28-7
- control (MDCR) 28-6
- data size (MDDSR) 28-11
- error status (MDESR) 28-10
- input FIFO (MDIF) 28-12
- message data size (MDMDS) 28-12
- message digest registers 0 (MDDx0) 28-12
- message digest registers 1 (MDDx1) 28-13
- mode (MDMR) 28-4
- status (MDSR) 28-8

### Memory map

- CCM 9-3
- chip select module 15-6
- clock module 7-8
- debug 31-5
- EMAC 4-6
- EPORT 14-2
- Ethernet 19-6
- GPIO 12-10
- I<sup>2</sup>C 26-8
- interrupt controller 13-4
- JTAG 32-5
- MDHA 28-3
- PIT 23-2
- power management 8-1
- PWM 22-2
- QSPI 24-9
- reset controller 10-2
- RNG 30-1
- SCM 11-2
- SKHA 29-6
- timers
  - DTIM 25-3
  - WDT 21-2
- UART modules 27-4
- USB 20-4

## P

### PLL

- charge pump/loop filter 7-22, 7-23
- lock detection 7-23
- loss-of-clock
  - alternate clock selection 7-25
  - detection 7-25
  - reset 7-25
  - stop mode 7-26
- loss-of-lock
  - conditions 7-24
  - reset 7-25
- multiplication factor divider (MFD) 7-23
- operation 7-21
  - normal mode 7-5
- phase and frequency detector (PFD) 7-22
- voltage control output (VCO) 7-23

### Power management

- low-power modes 8-4
    - doze 8-5
    - peripheral behavior
      - CCM 8-9
      - chip select module 8-7
      - clock module 8-10
      - core 8-6
      - debug 8-11
      - DMA controller 8-7
      - DTIM 8-8
      - EPORT 8-10
      - Ethernet 8-9
      - GPIO 8-9
      - I<sup>2</sup>C 8-7
      - interrupt controller 8-8
      - JTAG 8-11
      - PIT 8-10
      - QSPI 8-8
      - reset controller 8-9
      - SCM 8-6
      - SDRAM controller 8-6
      - SRAM 8-6
      - UART modules 8-7
      - WDT 8-10
    - run 8-5
    - stop 8-5
    - summary 8-11
    - wait 8-5
  - memory map 8-1
  - registers
    - low-power control (LPCR) 8-3
    - low-power interrupt control (LPICR) 8-2
- Program counter 3-3
- ### PWM
- channel timers 22-12–22-19
  - clock select 22-10–22-12

- memory map 22-2
- registers
  - center align enable (PWMCAE) 22-5
  - channel counter (PWMCNT $n$ ) 22-8
  - channel duty (PWMDTY $n$ ) 22-10
  - channel period (PWMPER $n$ ) 22-9
  - clock select (PWMCLK) 22-4
  - control (PWMCTL) 22-6
  - enable (PWME) 22-2
  - polarity (PWMPOL) 22-3
  - prescale clock select (PWMPRCLK) 22-4
  - scale A (PWMSCLA) 22-7
  - scale B (PWMSCLB) 22-7

## Q

### QSPI

- baud rate 24-6
- memory map 24-9
- operation
  - low-power modes 8-8
  - master mode 24-3
- programming example 24-16

### RAM

- command 24-6
- model 24-4
- receive 24-5
- transmit 24-5

### registers

- address (QAR) 24-14
- command RAM (QCR $n$ ) 24-14
- data (QDR) 24-14
- delay (QDLYR) 24-11
- interrupt (QIR) 24-12
- mode (QMR) 24-9
- wrap (QWR) 24-12

### Rx

- RAM 24-5

### signals

### Tx

- delays 24-7
- length 24-8
- RAM 24-5

## R

### Registers

#### cache

- access control 0–1 (ACR $n$ ) 3-8, 5-10
- control (CACR) 3-7, 5-7

#### CCM

- chip configuration (CCR) 9-4
- chip identification (CIR) 9-7
- reset configuration (RCON) 9-5

#### chip select module

- address (CSAR $n$ ) 15-7

- control (CSCR $n$ ) 15-9

- mask (CSMR $n$ ) 15-8

#### clock module

- synthesizer control (SYNCR) 7-8

- synthesizer status (SYNSR) 7-11

#### core

- address (A $n$ ) 3-3

- condition code (CCR) 3-4

- data (D $n$ ) 3-2

- program counter (PC) 3-3

- stack pointer (A7) 3-3, 3-7

- status register (SR) 3-6

- vector base (VBR) 3-7

#### debug

- address attribute trigger (AATR) 31-7

- address breakpoint (ABLR, ABHR) 31-8

- configuration/status (CSR) 31-9

- data breakpoint/mask (DBR, DBMR) 31-12

- program counter breakpoint/mask  
(PBR/PBMR) 31-13

- trigger definition (TDR) 31-14

#### DMA controller

- byte count (BCR $n$ ) 18-8

- control (DCR $n$ ) 18-10

- destination address (DAR $n$ ) 18-8

- request control (DMAREQC) 18-6

- source address (SAR $n$ ) 18-7

- status (DSR $n$ ) 18-9

#### EMAC

- mask (MASK) 4-11

- status (MACSR) 4-6

#### EPORT

- data direction (EPDDR) 14-4

- flag (EPFR) 14-6

- pin assignment (EPPAR) 14-3

- pin data (EPPDR) 14-6

- port data (EPDR) 14-5

- port interrupt enable (EPIER) 14-5

#### Ethernet

- control (ECR) 19-13

- descriptor group upper/lower address  
(GAUR/GALR) 19-24

- descriptor individual upper/lower  
(IAUR/IALR) 19-23

- descriptor individual upper/lower address  
(IAUR/IALR) 19-22

- FIFO receive bound (FRBR) 19-25

- FIFO receive start (FRSR) 19-26

- FIFO transmit FIFO watermark (TFWR) 19-25

- interrupt event (EIR) 19-9

- interrupt mask (EIMR) 19-10

- MIB control (MIBC) 19-17

- MII management frame (MMFR) 19-14

- MII speed control (MSCR) 19-16
- opcode/pause duration (OPD) 19-21
- physical address low (PALR<sub>n</sub>) 19-20
- physical address low/high (PALR, PAUR) 19-20
- receive buffer size (EMRBR) 19-28
- receive control (RCR) 19-18
- receive descriptor active (RDAR) 19-11
- receive descriptor ring start (ERDSR) 19-27
- transmit buffer descriptor ring start (ETSDR) 19-27
- transmit control (TCR) 19-19
- transmit descriptor active (TDAR) 19-12
- GPIO
  - pin assignment (PAR<sub>x</sub>) 12-21
  - port clear output data (PCLRR<sub>x</sub>) 12-19
  - port data direction (PDDR<sub>x</sub>) 12-14
  - port output data (PODR<sub>x</sub>) 12-11
  - port pin data/set data (PPDSR<sub>x</sub>) 12-16
- I<sup>2</sup>C
  - address (I2ADR) 26-8
  - control (I2CR) 26-10
  - data I/O (I2DR) 26-12
  - frequency divider (I2FDR) 26-9
  - status (I2SR) 26-11
- interrupt controller
  - interrupt acknowledge level and priority (IACKLPR<sub>n</sub>) 13-11, 1-9
  - interrupt control (ICR<sub>nx</sub>) 13-12
  - interrupt force high/low (INTFRCH<sub>n</sub>, INTFRCL<sub>n</sub>) 13-9
  - interrupt pending high/low (IPRH<sub>n</sub>, IPLR<sub>n</sub>) 13-6
  - interrupt request level (IRLR<sub>n</sub>) 13-11
  - mask high/low (IMRH<sub>n</sub>, n) 13-7
- JTAG
  - bypass 32-6
  - IDCODE 32-5
  - instruction shift (IR) 32-5
  - TEST\_CTRL 32-6
- MDHA
  - command (MDCMR) 28-7
  - control (MDCR) 28-6
  - data size (MDDSR) 28-11
  - error status (MDESR) 28-10
  - input FIFO (MDIF) 28-12
  - message data size (MDMDS) 28-12
  - message digest registers 0 (MDDx0) 28-12
  - message digest registers 1 (MDDx1) 28-13
  - mode (MDMR) 28-4
  - status (MDSR) 28-8
- power management
  - low-power control (LPCR) 8-3
  - low-power interrupt control (LPICR) 8-2
- PWM
  - center align enable (PWMCAE) 22-5
  - channel counter (PWCNT<sub>n</sub>) 22-8
  - channel duty (PWMDTY<sub>n</sub>) 22-10
  - channel period (PWMPER<sub>n</sub>) 22-9
  - clock select (PWMCLK) 22-4
  - control (PWMCTL) 22-6
  - enable (PWME) 22-2
  - polarity (PWMPOL) 22-3
  - prescale clock select (PWMPRCLK) 22-4
  - scale A (PWMSCLA) 22-7
  - scale B (PWMSCLB) 22-7
- QSPI
  - address (QAR) 24-14
  - command RAM (QCR<sub>n</sub>) 24-14
  - data (QDR) 24-14
  - delay (QDLYR) 24-11
  - interrupt (QIR) 24-12
  - mode (QMR) 24-9
  - wrap (QWR) 24-12
- reset controller
  - control (RCR) 10-2
  - status (RSR) 10-3
- RNG
  - control (RNGCR) 30-1
  - entropy (RNGER) 30-3
  - output FIFO (RNGOUT) 30-4
  - status (RNGSR) 30-2
- SCM
  - bus master park (MPARK) 11-11
  - core reset status (CRSR) 11-6
  - core watchdog control (CWCR) 11-7
  - core watchdog service (CWSR) 11-8
  - grouped peripheral access control (GPACR) 11-17
  - IPSBAR 11-3
  - master privilege (MPR) 11-15
  - peripheral access control (PACR<sub>n</sub>) 11-15
  - RAMBAR 3-8, 11-4
- SDRAM controller
  - configuration 1 (SDCFG1) 17-17
  - configuration 2 (SDCFG2) 17-19
  - control (SDCR) 17-15
  - mode/extended mode (SDMR) 17-14
- SKHA
  - command (SKCMR) 29-9
  - context (SKCR<sub>n</sub>) 29-15
  - control (SKCR) 29-8
  - data size (SKDSR) 29-13
  - error status (SKESR) 29-11
  - error status mask (SKESMR) 29-12
  - key data (SKKDR<sub>n</sub>) 29-14
  - key size (SKKSR) 29-13
  - mode (SKMR) 29-7
  - status (SKSR) 29-10
- SRAM
  - RAMBAR 6-2
- timers

- DTIM
  - capture (DTCR $n$ ) 25-8
  - counters (DTCN $n$ ) 25-8
  - event (DTER $n$ ) 25-6
  - mode (DTMR $n$ ) 25-4
  - reference (DTRR $n$ ) 25-7
- PIT
  - control and status (PCSR) 23-3
  - count (PCNTR) 23-5
  - modulus (PMR) 23-5
- WDT
  - control (WCR) 21-3
  - count (WCNTR) 21-4
  - modulus (WMR) 21-4
  - service (WSR) 21-4
- UART modules
  - auxiliary control (UACR $n$ ) 27-13
  - baud rate generator (UBG1 $n$ /UBG2 $n$ ) 27-15
  - clock select (UCSR $n$ ) 27-10
  - command (UCR $n$ ) 27-10
  - input port (UIP $n$ ) 27-16
  - input port change (UIPCR $n$ ) 27-13
  - interrupt status/mask (UISR $n$ /UIMR $n$ ) 27-14
  - mode 1 (UMR1 $n$ ) 27-5
  - mode 2 (UMR $n$ ) 27-7
  - output port command (UOP1 $n$ /UOP0 $n$ ) 27-16
  - receive buffers (URB $n$ ) 27-12
  - status (USR $n$ ) 27-8
  - transmit buffers (UTB $n$ ) 27-12
- USB
  - control (USB\_CTRL) 20-7
  - descriptor RAM address (USB\_DADR) 20-9
  - descriptor RAM/endpoint buffer data (USB\_DDAT) 20-10
  - endpoint  $n$  FIFO alarm (USB\_EP $n$ \_FALRM) 20-22
  - endpoint  $n$  FIFO control (USB\_EP $n$ \_FCTRL) 20-19
  - endpoint  $n$  FIFO data (USB\_EP $n$ \_FDAT) 20-17
  - endpoint  $n$  FIFO read pointer (USB\_EP $n$ \_FRDP) 20-23
  - endpoint  $n$  FIFO status (USB\_EP $n$ \_FSTAT) 20-18
  - endpoint  $n$  FIFO write pointer (USB\_EP $n$ \_FWRP) 20-24
  - endpoint  $n$  interrupt mask (USB\_EP $n$ \_MASK) 20-17
  - endpoint  $n$  interrupt status (USB\_EP $n$ \_INTR) 20-15
  - endpoint  $n$  last read frame pointer (USB\_EP $n$ \_LRFP) 20-21
  - endpoint  $n$  last write frame pointer (USB\_EP $n$ \_LWFP) 20-21
  - endpoint  $n$  status/control (USB\_EP $n$ \_STAT) 20-13
  - FIFO memory control (USB\_MCTL) 20-13
  - frame number and match (USB\_FRAME) 20-6
  - interrupt mask (USB\_MASK) 20-12
  - interrupt status (USB\_INTR) 20-10
  - specification/release number (USB\_SPEC) 20-6

- status (USB\_STAT) 20-7
- Reset controller
  - control flow 10-6
  - low-power modes 8-9
  - memory map 10-2
  - registers
    - control (RCR) 10-2
    - status (RSR) 10-3
  - requests
    - internal 10-8
    - synchronous 10-8
  - reset sources 10-4–10-6
  - signals
    - RESET 10-2
    - RSTOUT 10-2
  - status flags 10-9
- RNG
  - memory map 30-1
  - registers
    - control (RNGCR) 30-1
    - entropy (RNGER) 30-3
    - output FIFO (RNGOUT) 30-4
    - status (RNGSR) 30-2

## S

- SACU 11-13
- SCM
  - low-power modes 8-6
  - memory map 11-2
  - registers
    - bus master park (MPARK) 11-11
    - core reset status (CRSR) 11-6
    - core watchdog control (CWCR) 11-7
    - core watchdog service (CWSR) 11-8
    - grouped peripheral access control (GPACR) 11-17
    - IPSBAR 11-3
    - master privilege (MPR) 11-15
    - peripheral access control (PACR $n$ ) 11-15
    - RAMBAR 3-8, 11-4
- SACU 11-13
- SDRAM controller
  - block diagram 17-2
  - commands
    - LMR, LEMR 17-10
    - PALL 17-10
    - PDWN 17-12
    - READ 17-9
    - REF 17-12
    - SREF 17-12
    - WRITE 17-10
  - configuration 17-4
  - example 17-23–17-33
  - initialization 17-13, 17-33
  - interface configuration 17-23–17-32



- operation
  - low-power modes 8-6
- page management 17-21
- registers
  - configuration 1 (SDCFG1) 17-17
  - configuration 2 (SDCFG2) 17-19
  - control (SDCR) 17-15
  - mode/extended mode (SDMR) 17-14
- transfer size 17-22
- Signals
  - block diagram 2-2
  - bus 16-1
  - CCM
    - CLKMOD1–0 9-2
    - RCON 9-2
  - chip select module
    - byte strobes ( $\overline{BS}3-0$ ) 15-2
    - chip select ( $\overline{CS}7-0$ ) 15-1
    - output enable ( $\overline{OE}$ ) 15-1
  - clock module
    - CLKMOD1–0 7-7
    - CLKOUT 7-7
    - EXTAL 7-7
    - RSTOUT 7-7
    - XTAL 7-7
  - debug
    - breakpoint ( $\overline{BKPT}$ ) 31-2
    - debug data (DDATA3–0) 31-2
    - development serial clock (DSCLK) 31-2
    - development serial input (DSI) 31-2
    - development serial output (DSO) 31-2
    - processor status (PST3–0) 31-2
    - PSTCLK 31-2
  - JTAG
    - JTAG\_EN 32-3
    - TCLK 32-4
    - test data input/development serial input (TDI/DSI) 32-4
    - test data output/development serial output (TDO/DSO) 32-5
    - test mode select/breakpoint (TMS/ $\overline{BKPT}$ ) 32-4
    - test reset/development serial clock ( $\overline{TRST}$ /DSCLK) 32-4
  - QSPI
    - summary 24-2
  - reset controller
    - RESET 10-2
    - RSTOUT 10-2
  - USB
    - USB clock (USB\_CLK) 20-3
    - USB receive data (USB\_RXD) 20-4
    - USB receive input (USB\_RN) 20-4
    - USB receive input (USB\_RP) 20-4
    - USB speed (USB\_SPEED) 20-3
    - USB suspended (USB\_SUSP) 20-4
    - USB transmit enable (USB\_TXEN) 20-4
    - USB transmit output (USB\_TN) 20-4
    - USB transmit output (USB\_TP) 20-4
- SKHA
  - FIFO 29-14, 29-17
  - logic 29-17
  - memory map 29-6
  - operation
    - AES 29-3
    - CBC 29-4
    - context switch 29-20
    - CTR 29-5
    - DES, 3DES 29-1
    - ECB 29-3
    - general 29-19
  - registers
    - command (SKCMR) 29-9
    - context (SKCR $n$ ) 29-15
    - control (SKCR) 29-8
    - data size (SKDSR) 29-13
    - error status (SKESR) 29-11
    - error status mask (SKESMR) 29-12
    - key data (SKKDR $n$ ) 29-14
    - key size (SKKSR) 29-13
    - mode (SKMR) 29-7
    - status (SKSR) 29-10
- SRAM
  - cache, interaction 5-3
  - initialization 6-4
  - operation
    - low-power modes 8-6
  - power management 6-5
  - registers
    - RAMBAR 6-2
- Stack pointer 3-3, 3-7
- System clock 7-15

## T

- TAP controller 32-7
- Timers
  - DTIM
    - capture mode 25-3
    - code example 25-10
    - memory map 25-3
    - operation
      - general 25-9
    - output mode 25-3
    - prescaler 25-2
    - reference compare 25-3
    - registers
      - capture (DTCR $n$ ) 25-8
      - counters (DTCN $n$ ) 25-8
      - event (DTER $n$ ) 25-6

- mode (DTMR $n$ ) 25-4
- reference (DTRR $n$ ) 25-7
- time-out values 25-11
- PIT
  - block diagram 23-1
  - interrupts 23-7
  - memory map 23-2
  - operation
    - free-running 23-6
    - low-power modes 8-10, 23-2
    - set-and-forget 23-6
  - registers
    - control and status (PCSR) 23-3
    - count (PCNTR) 23-5
    - modulus (PMR) 23-5
  - timeout 23-7
- WDT
  - memory map 21-2
  - operation
    - low-power 8-10, 21-1
  - registers
    - control (WCR) 21-3
    - count (WCNTR) 21-4
    - modulus (WMR) 21-4
    - service (WSR) 21-4

## U

UART modules

- clock select registers (UCSR $n$ ) 27-10
- clock source
  - baud rates 27-18
  - divider 27-17
  - external 27-19
- command registers (UCR $n$ ) 27-10
- core interrupts 27-28
- DMA service 27-28
- FIFO stack 27-23
- initialization 27-30
- input port change (UIPCR $n$ ) 27-13
- memory map 27-4
- operation
  - looping modes
    - automatic echo 27-24
    - local loop-back 27-24
    - remote loop-back 27-25
  - low-power modes 8-7
  - multidrop mode 27-25
  - receiver 27-21
  - transmitter 27-19
- registers
  - auxiliary control (UACR $n$ ) 27-13
  - baud rate generator (UBG1 $n$ /UBG2 $n$ ) 27-15
  - input port (UIP $n$ ) 27-16
  - interrupt status/mask (UISR $n$ /UIMR $n$ ) 27-14

- mode 1 (UMR1 $n$ ) 27-5
- mode 2 (UMR2 $n$ ) 27-7
- output port command (UOP1 $n$ /UOP0 $n$ ) 27-16
- receive buffers (URB $n$ ) 27-12
- status (USR $n$ ) 27-8
- transmit buffers (UTB $n$ ) 27-12

USB

- interrupts 20-27–20-31
- memory map 20-4
- packets 20-38–20-39
- registers
  - control (USB\_CTRL) 20-7
  - descriptor RAM address (USB\_DADR) 20-9
  - descriptor RAM/endpoint buffer data (USB\_DDAT) 20-10
  - endpoint  $n$  FIFO alarm (USB\_EP $n$ \_FALRM) 20-22
  - endpoint  $n$  FIFO control (USB\_EP $n$ \_FCTRL) 20-19
  - endpoint  $n$  FIFO data (USB\_EP $n$ \_FDAT) 20-17
  - endpoint  $n$  FIFO read pointer (USB\_EP $n$ \_FRDP) 20-23
  - endpoint  $n$  FIFO status (USB\_EP $n$ \_FSTAT) 20-18
  - endpoint  $n$  FIFO write pointer (USB\_EP $n$ \_FWRP) 20-24
  - endpoint  $n$  interrupt mask (USB\_EP $n$ \_MASK) 20-17
  - endpoint  $n$  interrupt status (USB\_EP $n$ \_INTR) 20-15
  - endpoint  $n$  last read frame pointer (USB\_EP $n$ \_LRFP) 20-21
  - endpoint  $n$  last write frame pointer (USB\_EP $n$ \_LWFP) 20-21
  - endpoint  $n$  status/control (USB\_EP $n$ \_STAT) 20-13
  - FIFO memory control (USB\_MCTL) 20-13
  - frame number and match (USB\_FRAME) 20-6
  - interrupt mask (USB\_MASK) 20-12
  - interrupt status (USB\_INTR) 20-10
  - specification/release number (USB\_SPEC) 20-6
  - status (USB\_STAT) 20-7
- reset 20-25–20-26
- signals
  - USB clock (USB\_CLK) 20-3
  - USB receive data (USB\_RXD) 20-4
  - USB receive input (USB\_RN) 20-4
  - USB receive input (USB\_RP) 20-4
  - USB speed (USB\_SPEED) 20-3
  - USB suspended (USB\_SUSP) 20-4
  - USB transmit enable (USB\_TXEN) 20-4
  - USB transmit output (USB\_TN) 20-4
  - USB transmit output (USB\_TP) 20-4

Overview	1
Signal Descriptions	2
ColdFire Core	3
Enhanced Multiply-Accumulate Unit (EMAC)	4
Cache	5
Static RAM (SRAM)	6
Clock Module	7
Power Management	8
Chip Configuration Module (CCM)	9
Reset Controller Module	10
System Control Module (SCM)	11
General Purpose I/O Module	12
Interrupt Controller Modules	13
Edge Port Module (EPORT)	14
Chip Select Module	15
External Interface Module (EIM)	16
Synchronous DRAM Controller	17
DMA Controller Module	18
Fast Ethernet Controller (FEC)	19
Universal Serial Bus Device (USB)	20
Watchdog Timer Module	21
PWM Module	22
Programmable Interrupt Timers (PITs)	23
DMA Timers	24
Queued Serial Peripheral Interface (QSPI)	25
UART Modules	26
I <sup>2</sup> C interface	27
Message Digest Hardware Accelerator (MDHA)	28
Random Number Generator (RNG)	29
Symmetric Key Hardware Accelerator (SKHA)	30
IEEE 1149.1 Test Access Port (JTAG)	31
Debug Support	32
Register Memory Map Quick Reference	A
Index	IND

1	Overview
2	Signal Descriptions
3	ColdFire Core
4	Enhanced Multiply-Accumulate Unit (EMAC)
5	Cache
6	Static RAM (SRAM)
7	Clock Module
8	Power Management
9	Chip Configuration Module (CCM)
10	Reset Controller Module
11	System Control Module (SCM)
12	General Purpose I/O Module
13	Interrupt Controller Modules
14	Edge Port Module (EPORT)
15	Chip Select Module
16	External Interface Module (EIM)
17	Synchronous DRAM Controller
18	DMA Controller Module
19	Fast Ethernet Controller (FEC)
20	Universal Serial Bus Device (USB)
21	Watchdog Timer Module
22	PWM Module
23	Programmable Interrupt Timers (PITs)
24	DMA Timers
25	Queued Serial Peripheral Interface (QSPI)
26	UART Modules
27	I <sup>2</sup> C interface
28	Message Digest Hardware Accelerator (MDHA)
29	Random Number Generator (RNG)
30	Symmetric Key Hardware Accelerator (SKHA)
31	IEEE 1149.1 Test Access Port (JTAG)
32	Debug Support
A	Register Memory Map Quick Reference
IND	Index