

# **MCF51CN128 ColdFire® Integrated Microcontroller Reference Manual**

**Devices Supported:**  
**MCF51CN128**

Document Number: MCF51CN128RM  
Rev. 6  
12/2009

## **How to Reach Us:**

**Home Page:**  
<http://www.freescale.com>

**E-mail:**  
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

### **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

### **Japan:**

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

### **Asia/Pacific:**

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

### **For Literature Requests Only:**

Freescale Semiconductor Literature Distribution Center  
1-800-441-2447 or 303-675-2140  
Fax: 303-675-2150  
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. 2009. All rights reserved.

MCF51CN128RM  
Rev. 6  
12/2009

## Chapter 1 Device Overview

1.1	The MCF51CN128 Series Microcontrollers .....	1-1
1.1.1	Definition .....	1-1
1.1.2	MCF51CN128 Series Devices .....	1-1
1.1.3	MCF51CN128 Series Device Comparison .....	1-2
1.2	MCF51CN128 Series Block Diagram .....	1-3
1.2.1	Block Diagram .....	1-3
1.2.2	Functional Units .....	1-5
1.2.3	Module Versions .....	1-6
1.3	V1 ColdFire Core .....	1-7
1.3.1	User Programming Model .....	1-7
1.3.2	Supervisor Programming Model .....	1-7
1.4	System Clock Generation and Distribution .....	1-8
1.4.1	Clock Distribution Diagram .....	1-8
1.4.2	System Clocks .....	1-9
1.4.3	Clock Gating .....	1-9
1.4.4	MCG Modes of Operation .....	1-10
1.4.5	MCG Mode State Diagram .....	1-12

## Chapter 2 Pins and Connections

2.1	Package Pin Assignments .....	2-1
2.1.1	Pinout: 80-Pin LQFP .....	2-1
2.1.2	Pinout: 64-Pin LQFP .....	2-3
2.1.3	Pinout: 48-Pin QFN .....	2-4
2.2	Pin Assignment Tables .....	2-5
2.2.1	Peripheral Pinout Summary .....	2-8
2.3	Pin Mux Controls .....	2-18
2.4	Basic System Connections .....	2-30
2.4.1	Power .....	2-32
2.4.2	Oscillator .....	2-32
2.4.3	RESET/PTC3 .....	2-33
2.4.4	IRQ .....	2-33
2.4.5	Background / Mode Select (BKGD/MS) .....	2-33
2.4.6	ADC Reference Pins ( $V_{REFH}$ , $V_{REFL}$ ) .....	2-34
2.4.7	General-Purpose I/O and Peripheral Ports .....	2-34

## Chapter 3 Modes of Operation

3.1	Introduction .....	3-1
3.2	Features .....	3-1
3.3	Overview .....	3-2
3.4	Debug Mode .....	3-6

3.5	Secure Mode .....	3-6
3.6	Run Modes .....	3-7
	3.6.1 Run Mode .....	3-7
	3.6.2 Low-Power Run Mode (LPrun) .....	3-7
	3.6.2.1 BDM in Low-Power Run Mode .....	3-7
3.7	Wait Modes .....	3-8
	3.7.1 Wait Mode .....	3-8
	3.7.2 Low-Power Wait Mode (LPwait) .....	3-8
	3.7.2.1 BDM in Low-Power Wait Mode .....	3-8
3.8	Stop Modes .....	3-9
	3.8.1 Stop2 Mode .....	3-9
	3.8.1.1 Oscillator Considerations for Stop2 .....	3-10
	3.8.2 Stop3 Mode .....	3-10
	3.8.3 Stop4 Mode .....	3-11
3.9	On-Chip Peripheral Modules in Stop and Low-Power Modes .....	3-11

## Chapter 4 Memory

4.1	MCF51CN128 Series Memory Map .....	4-1
4.2	Detailed Register Addresses and Bit Assignments .....	4-5
	4.2.1 Flash Module Reserved Memory Locations .....	4-22
4.3	RAM .....	4-23
4.4	Flash Memory .....	4-24
	4.4.1 Features .....	4-24
	4.4.2 Register Descriptions .....	4-25
	4.4.2.1 Flash Clock Divider Register (FCDIV) .....	4-25
	4.4.2.2 Flash Options Register (FOPT and NVOPT) .....	4-26
	4.4.2.3 Flash Configuration Register (FCNFG) .....	4-27
	4.4.2.4 Flash Protection Register (FPROT and NVPROT) .....	4-27
	4.4.2.5 Flash Status Register (FSTAT) .....	4-29
	4.4.2.6 Flash Command Register (FCMD) .....	4-30
4.5	Security .....	4-31

## Chapter 5 Resets, Interrupts, and General System Control

5.1	Introduction .....	5-1
5.2	Features .....	5-1
5.3	Microcontroller Reset .....	5-1
	5.3.1 Computer Operating Properly (COP) Watchdog .....	5-2
	5.3.2 Illegal Opcode Detect (IOP) .....	5-2
	5.3.3 Illegal Address Detect (ILAD) .....	5-3
5.4	Interrupts & Exceptions .....	5-3
	5.4.1 RESET/PTC3 .....	5-4
	5.4.2 External Interrupt Request (IRQ) Pin .....	5-4

5.4.2.1	Pin Configuration Options .....	5-4
5.4.2.2	Edge and Level Sensitivity .....	5-5
5.4.3	Interrupt Vectors, Sources, and Local Masks .....	5-5
5.5	Low-Voltage Detect (LVD) System .....	5-10
5.5.1	Power-On Reset Operation .....	5-10
5.5.2	LVD Reset Operation .....	5-10
5.5.3	LVD Interrupt Operation .....	5-10
5.5.4	Low-Voltage Warning (LVW) Interrupt Operation .....	5-10
5.6	Peripheral Clock Gating .....	5-11
5.7	Reset, Interrupt, and System Control Registers and Control Bits .....	5-11
5.7.1	Interrupt Pin Request Status and Control Register (IRQSC) .....	5-11
5.7.2	System Reset Status Register (SRS) .....	5-12
5.7.3	System Options 1 Register (SOPT1) .....	5-14
5.7.4	System Options 2 Register (SOPT2) .....	5-15
5.7.5	System Options Register 3 (SOPT3) .....	5-16
5.7.6	System Device Identification Register (SDIDH, SDIDL) .....	5-16
5.7.7	System Power Management Status and Control 1 Register (SPMSC1) .....	5-17
5.7.8	System Power Management Status and Control 2 Register (SPMSC2) .....	5-18
5.7.9	System Power Management Status and Control 3 Register (SPMSC3) .....	5-19
5.7.10	System Clock Gating Control 1 Register (SCGC1) .....	5-21
5.7.11	System Clock Gating Control 2 Register (SCGC2) .....	5-22
5.7.12	System Clock Gating Control 3 Register (SCGC3) .....	5-23
5.7.13	System Clock Gating Control 4 Register (SCGC4) .....	5-23
5.7.14	SIM Internal Peripheral Select Register (SIMIPS) .....	5-24

## Chapter 6

### Multipurpose Clock Generator (MCG)

6.1	Introduction .....	6-1
6.1.1	Features .....	6-2
6.1.2	Modes of Operation .....	6-4
6.2	External Signal Description .....	6-4
6.3	Register Definition .....	6-4
6.3.1	MCG Control Register 1 (MCGC1) .....	6-4
6.3.2	MCG Control Register 2 (MCGC2) .....	6-6
6.3.3	MCG Trim Register (MCGTRM) .....	6-7
6.3.4	MCG Status and Control Register (MCGSC) .....	6-8
6.3.5	MCG Control Register 3 (MCGC3) .....	6-9
6.3.6	MCG Control Register 4 (MCGC4) .....	6-10
6.4	Functional Description .....	6-11
6.4.1	MCG Modes of Operation .....	6-11
6.4.2	MCG Mode State Diagram .....	6-13
6.4.3	Mode Switching .....	6-14
6.4.4	Bus Frequency Divider .....	6-15
6.4.5	Low Power Bit Usage .....	6-15
6.4.6	Internal Reference Clock .....	6-15

6.4.7	External Reference Clock .....	6-15
6.4.8	Fixed Frequency Clock .....	6-16
6.5	Initialization / Application Information .....	6-16
6.5.1	MCG Module Initialization Sequence .....	6-16
6.5.1.1	Initializing the MCG .....	6-16
6.5.2	Using a 32.768 kHz Reference .....	6-18
6.5.3	MCG Mode Switching .....	6-18
6.5.3.1	Example 1: Moving from FEI to PEE Mode: External Crystal = 8 MHz, Bus Frequency = 16 MHz .....	6-19
6.5.3.2	Example 2: Moving from PEE to BLPI Mode: Bus Frequency =16 kHz .....	6-23
6.5.3.3	Example 3: Moving from BLPI to FEE Mode: External Crystal = 8 MHz, Bus Frequency = 16 MHz .....	6-26
6.5.4	Calibrating the Internal Reference Clock (IRC) .....	6-27
6.5.4.1	Example 4: Internal Reference Clock Trim .....	6-28

## Chapter 7 ColdFire Core

7.1	Introduction .....	7-1
7.1.1	Overview .....	7-1
7.2	Memory Map/Register Description .....	7-2
7.2.1	Data Registers (D0–D7) .....	7-3
7.2.2	Address Registers (A0–A6) .....	7-4
7.2.3	Supervisor/User Stack Pointers (A7 and OTHER_A7) .....	7-4
7.2.4	Condition Code Register (CCR) .....	7-5
7.2.5	Program Counter (PC) .....	7-6
7.2.6	Vector Base Register (VBR) .....	7-6
7.2.7	CPU Configuration Register (CPUCR) .....	7-7
7.2.8	Status Register (SR) .....	7-8
7.3	Functional Description .....	7-9
7.3.1	Instruction Set Architecture (ISA_C) .....	7-9
7.3.2	Exception Processing Overview .....	7-10
7.3.2.1	Exception Stack Frame Definition .....	7-12
7.3.3	Processor Exceptions .....	7-14
7.3.3.1	Access Error Exception .....	7-14
7.3.3.2	Address Error Exception .....	7-14
7.3.3.3	Illegal Instruction Exception .....	7-15
7.3.3.4	Privilege Violation .....	7-16
7.3.3.5	Trace Exception .....	7-16
7.3.3.6	Unimplemented Line-A Opcode .....	7-17
7.3.3.7	Unimplemented Line-F Opcode .....	7-17
7.3.3.8	Debug Interrupt .....	7-17
7.3.3.9	RTE and Format Error Exception .....	7-17
7.3.3.10	TRAP Instruction Exception .....	7-18
7.3.3.11	Unsupported Instruction Exception .....	7-18
7.3.3.12	Interrupt Exception .....	7-18

7.3.3.13	Fault-on-Fault Halt .....	7-18
7.3.3.14	Reset Exception .....	7-19
7.3.4	Instruction Execution Timing .....	7-22
7.3.4.1	Timing Assumptions .....	7-22
7.3.4.2	MOVE Instruction Execution Times .....	7-23
7.3.4.3	Standard One Operand Instruction Execution Times .....	7-24
7.3.4.4	Standard Two Operand Instruction Execution Times .....	7-25
7.3.4.5	Miscellaneous Instruction Execution Times .....	7-26
7.3.4.6	Branch Instruction Execution Times .....	7-27

## Chapter 8 Interrupt Controller (CF1\_INTC)

8.1	Introduction .....	8-1
8.1.1	Overview .....	8-2
8.1.2	Features .....	8-8
8.1.3	Modes of Operation .....	8-9
8.2	External Signal Description .....	8-9
8.3	Memory Map/Register Definition .....	8-9
8.3.1	Interrupt OR Mask Register (INTC_ORMR) .....	8-10
8.3.2	Force Interrupt Register (INTC_FRC) .....	8-11
8.3.3	INTC Programmable Level 6, Priority {7,6} Registers (INTC_PL6P{7,6}) .....	8-12
8.3.4	INTC Wakeup Control Register (INTC_WCR) .....	8-13
8.3.5	INTC Set Interrupt Force Register (INTC_SFRC) .....	8-14
8.3.6	INTC Clear Interrupt Force Register (INTC_CFRC) .....	8-15
8.3.7	INTC Software and Level- <i>n</i> IACK Registers ( <i>n</i> = 1,2,3,...,7) .....	8-16
8.4	Functional Description .....	8-17
8.4.1	Handling of Non-Maskable Level 7 Interrupt Requests .....	8-17
8.5	Initialization Information .....	8-17
8.6	Application Information .....	8-17
8.6.1	Emulation of the HCS08's 1-Level IRQ Handling .....	8-17
8.6.2	Using INTC_PL6P{7,6} Registers .....	8-18
8.6.3	More on Software IACKs .....	8-19

## Chapter 9 Parallel Input/Output Control

9.1	Overview of MCF51CN128 I/O Functions .....	9-1
9.1.1	Summary .....	9-1
9.1.2	Ports Implemented via Rapid GPIO .....	9-1
9.1.3	Keyboard Functions .....	9-2
9.1.4	Port Mux Control .....	9-2
9.1.5	Special Cases .....	9-2
9.1.5.1	Pull-Up Resistors .....	9-2
9.1.5.2	Port J .....	9-3
9.1.5.3	<u>RESET</u> .....	9-3

9.2	Pin Controls .....	9-4
9.2.1	Pin Controls Overview .....	9-4
9.2.2	Pin Controls Programming Model .....	9-4
9.2.2.1	Port x Pull Enable Register (PTxPE) .....	9-5
9.2.2.2	Port x Slew Rate Enable Register (PTxSE) .....	9-6
9.2.2.3	Port x Drive Strength Selection Register (PTxDS) .....	9-6
9.2.2.4	Port x Input Filter Enable Register (PTxIFE) .....	9-7
9.3	Standard GPIO .....	9-7
9.3.1	GPIO Overview .....	9-7
9.3.2	GPIO Programming Model .....	9-8
9.3.2.1	Port x Data Register (PTxD) .....	9-9
9.3.2.2	Port x Data Direction Register (PTxDD) .....	9-9
9.4	V1 ColdFire Rapid GPIO Functionality .....	9-10
9.5	Keyboard Interrupts .....	9-10
9.5.1	Keyboard Functional Considerations .....	9-10
9.5.1.1	Edge Only Sensitivity .....	9-10
9.5.1.2	Edge and Level Sensitivity .....	9-11
9.5.1.3	Pull-up/Pull-down Resistors .....	9-11
9.5.1.4	Keyboard Interrupt Initialization .....	9-11
9.5.2	Keyboard Programming Model .....	9-11
9.5.2.1	KBI <sub>x</sub> Interrupt Status and Control Register (KBI <sub>x</sub> SC) .....	9-12
9.5.2.2	KBI <sub>x</sub> Interrupt Pin Select Register (KBI <sub>x</sub> PE) .....	9-12
9.5.2.3	KBI <sub>x</sub> Interrupt Edge Select Register (KBI <sub>x</sub> ES) .....	9-13
9.6	Pin Behavior in Stop Modes .....	9-13

## Chapter 10

### Rapid GPIO (RGPIO)

10.1	Introduction .....	10-1
10.1.1	Overview .....	10-1
10.1.2	Features .....	10-2
10.1.3	Modes of Operation .....	10-3
10.2	External Signal Description .....	10-3
10.2.1	Overview .....	10-3
10.2.2	Detailed Signal Descriptions .....	10-3
10.3	Memory Map/Register Definition .....	10-4
10.3.1	RGPIODirection (RGPIO_DIR) .....	10-5
10.3.2	RGPIOData (RGPIO_DATA) .....	10-5
10.3.3	RGPIOPinEnable (RGPIO_ENB) .....	10-6
10.3.4	RGPIOClearData (RGPIO_CLR) .....	10-6
10.3.5	RGPIOSetData (RGPIO_SET) .....	10-7
10.3.6	RGPIOToggleData (RGPIO_TOG) .....	10-7
10.4	Functional Description .....	10-8
10.5	Initialization Information .....	10-8
10.6	Application Information .....	10-8
10.6.1	Application 1: Simple Square-Wave Generation .....	10-9

## Chapter 11 Mini-FlexBus

11.1	Introduction .....	11-1
11.1.1	Overview .....	11-1
11.1.2	Features .....	11-2
11.1.3	Modes of Operation .....	11-2
11.2	External Signals .....	11-2
11.2.1	Address and Data Buses (FB_A[19:0], FB_D[7:0], FB_AD[:0]) .....	11-2
11.2.1	19 .....	11-2
11.2.2	Chip Selects ( $\overline{FB\_CS[1:0]}$ ) .....	11-3
11.2.3	Output Enable ( $\overline{FB\_OE}$ ) .....	11-3
11.2.4	Read/Write (FB_R/W) .....	11-3
11.2.5	Address Latch Enable (FB_ALE) .....	11-3
11.3	Memory Map/Register Definition .....	11-3
11.3.1	Chip-Select Address Registers (CSAR0 – CSAR1) .....	11-4
11.3.2	Chip-Select Mask Registers (CSMR0 – CSMR1) .....	11-4
11.3.3	Chip-Select Control Registers (CSCR0 – CSCR1) .....	11-5
11.4	Functional Description .....	11-7
11.4.1	Chip-Select Operation .....	11-7
11.4.1.1	General Chip-Select Operation .....	11-7
11.4.1.2	8- and 16-Bit Port Sizing .....	11-7
11.4.2	Data Transfer Operation .....	11-8
11.4.3	Data Byte Alignment and Physical Connections .....	11-8
11.4.4	Address/Data Bus Multiplexing .....	11-9
11.4.5	Bus Cycle Execution .....	11-9
11.4.5.1	Data Transfer Cycle States .....	11-10
11.4.6	Mini-FlexBus Timing Examples .....	11-10
11.4.6.1	Basic Read Bus Cycle .....	11-10
11.4.6.2	Basic Write Bus Cycle .....	11-12
11.4.6.3	Bus Cycle Sizing .....	11-13
11.4.6.4	Timing Variations .....	11-16
11.4.7	Bus Errors .....	11-21

## Chapter 12 Real-Time Counter (RTC)

12.1	Introduction .....	12-1
12.1.1	Features .....	12-2
12.1.2	Modes of Operation .....	12-2
12.1.2.1	Wait Mode .....	12-2
12.1.2.2	Stop Modes .....	12-2
12.1.2.3	Active Background Mode .....	12-2
12.1.3	Block Diagram .....	12-3

12.2	External Signal Description .....	12-3
12.3	Register Definition .....	12-3
12.3.1	RTC Status and Control Register (RTCSC) .....	12-4
12.3.2	RTC Counter Register (RTCCNT) .....	12-5
12.3.3	RTC Modulo Register (RTCMOD) .....	12-5
12.4	Functional Description .....	12-5
12.4.1	RTC Operation Example .....	12-6
12.5	Initialization/Application Information .....	12-7

## Chapter 13 Serial Communication Interface (SCI)

13.1	Introduction .....	13-1
13.1.1	Module Block Diagram .....	13-1
13.1.2	Features .....	13-3
13.1.3	Modes of Operation .....	13-3
13.1.4	Block Diagram .....	13-4
13.2	Register Definition .....	13-6
13.2.1	SCI Baud Rate Registers (SCIxBDH, SCIxBDL) .....	13-6
13.2.2	SCI Control Register 1 (SCIxC1) .....	13-7
13.2.3	SCI Control Register 2 (SCIxC2) .....	13-8
13.2.4	SCI Status Register 1 (SCIxS1) .....	13-9
13.2.5	SCI Status Register 2 (SCIxS2) .....	13-11
13.2.6	SCI Control Register 3 (SCIxC3) .....	13-12
13.2.7	SCI Data Register (SCIxD) .....	13-13
13.3	Functional Description .....	13-13
13.3.1	Baud Rate Generation .....	13-13
13.3.2	Transmitter Functional Description .....	13-14
13.3.2.1	Send Break and Queued Idle .....	13-15
13.3.3	Receiver Functional Description .....	13-15
13.3.3.1	Data Sampling Technique .....	13-16
13.3.3.2	Receiver Wakeup Operation .....	13-16
13.3.4	Interrupts and Status Flags .....	13-17
13.3.5	Additional SCI Functions .....	13-18
13.3.5.1	8- and 9-Bit Data Modes .....	13-18
13.3.5.2	Stop Mode Operation .....	13-19
13.3.5.3	Loop Mode .....	13-19
13.3.5.4	Single-Wire Operation .....	13-19

## Chapter 14 Serial Peripheral Interface (SPI)

14.1	Introduction .....	14-1
14.1.1	Features .....	14-2
14.1.2	Block Diagrams .....	14-2
14.1.2.1	SPI System Block Diagram .....	14-2

14.1.2.2 SPI Module Block Diagram .....	14-3
14.1.3 SPI Baud Rate Generation .....	14-4
14.2 External Signal Description .....	14-5
14.2.1 SPSCK — SPI Serial Clock .....	14-5
14.2.2 MOSI — Master Data Out, Slave Data In .....	14-5
14.2.3 MISO — Master Data In, Slave Data Out .....	14-5
14.2.4 SS — Slave Select .....	14-5
14.3 Modes of Operation .....	14-6
14.3.1 SPI in Stop Modes .....	14-6
14.4 Register Definition .....	14-6
14.4.1 SPI Control Register 1 (SPIxC1) .....	14-6
14.4.2 SPI Control Register 2 (SPIxC2) .....	14-7
14.4.3 SPI Baud Rate Register (SPIxBR) .....	14-8
14.4.4 SPI Status Register (SPIxS) .....	14-9
14.4.5 SPI Data Register (SPIxD) .....	14-10
14.5 Functional Description .....	14-11
14.5.1 SPI Clock Formats .....	14-11
14.5.2 SPI Interrupts .....	14-13
14.5.3 Mode Fault Detection .....	14-14

## Chapter 15 Analog-to-Digital Converter (ADC12)

15.1 Introduction .....	15-1
15.1.1 ADC Clock Gating .....	15-1
15.1.2 Module Configurations .....	15-1
15.1.2.1 Channel Assignments .....	15-1
15.1.2.2 Alternate Clock .....	15-2
15.1.2.3 Hardware Trigger .....	15-3
15.1.2.4 Temperature Sensor .....	15-3
15.1.3 Features .....	15-4
15.1.4 Block Diagram .....	15-4
15.2 External Signal Description .....	15-5
15.2.1 Analog Power (V <sub>DDAD</sub> ) .....	15-6
15.2.2 Analog Ground (V <sub>SSAD</sub> ) .....	15-6
15.2.3 Voltage Reference High (V <sub>REFH</sub> ) .....	15-6
15.2.4 Voltage Reference Low (V <sub>REFL</sub> ) .....	15-6
15.2.5 Analog Channel Inputs (AD <sub>x</sub> ) .....	15-6
15.3 Register Definition .....	15-6
15.3.1 Status and Control Register 1 (ADCSC1) .....	15-6
15.3.2 Status and Control Register 2 (ADCSC2) .....	15-8
15.3.3 Data Result High Register (ADC RH) .....	15-8
15.3.4 Data Result Low Register (ADC RL) .....	15-9
15.3.5 Compare Value High Register (ADCCVH) .....	15-9
15.3.6 Compare Value Low Register (ADCCVL) .....	15-10
15.3.7 Configuration Register (ADCCFG) .....	15-10

15.4	Functional Description . . . . .	15-11
15.4.1	Clock Select and Divide Control . . . . .	15-12
15.4.2	Input Select and Pin Control . . . . .	15-12
15.4.3	Hardware Trigger . . . . .	15-12
15.4.4	Conversion Control . . . . .	15-13
15.4.4.1	Initiating Conversions . . . . .	15-13
15.4.4.2	Completing Conversions . . . . .	15-13
15.4.4.3	Aborting Conversions . . . . .	15-13
15.4.4.4	Power Control . . . . .	15-14
15.4.4.5	Sample Time and Total Conversion Time . . . . .	15-14
15.4.5	Automatic Compare Function . . . . .	15-15
15.4.6	MCU Wait Mode Operation . . . . .	15-15
15.4.7	MCU Stop3 Mode Operation . . . . .	15-16
15.4.7.1	Stop3 Mode With ADACK Disabled . . . . .	15-16
15.4.7.2	Stop3 Mode With ADACK Enabled . . . . .	15-16
15.4.8	MCU Stop2 Mode Operation . . . . .	15-17
15.5	Initialization Information . . . . .	15-17
15.5.1	ADC Module Initialization Example . . . . .	15-17
15.5.1.1	Initialization Sequence . . . . .	15-17
15.5.1.2	Pseudo-Code Example . . . . .	15-17
15.6	Application Information . . . . .	15-19
15.6.1	External Pins and Routing . . . . .	15-19
15.6.1.1	Analog Supply Pins . . . . .	15-19
15.6.1.2	Analog Reference Pins . . . . .	15-19
15.6.1.3	Analog Input Pins . . . . .	15-20
15.6.2	Sources of Error . . . . .	15-20
15.6.2.1	Sampling Error . . . . .	15-20
15.6.2.2	Pin Leakage Error . . . . .	15-20
15.6.2.3	Noise-Induced Errors . . . . .	15-21
15.6.2.4	Code Width and Quantization Error . . . . .	15-21
15.6.2.5	Linearity Errors . . . . .	15-22
15.6.2.6	Code Jitter, Non-Monotonicity, and Missing Codes . . . . .	15-22

## Chapter 16

### Fast Ethernet Controller (FEC)

16.1	Introduction . . . . .	16-1
16.1.1	Overview . . . . .	16-1
16.1.2	Block Diagram . . . . .	16-2
16.1.3	Features . . . . .	16-3
16.2	Modes of Operation . . . . .	16-3
16.2.1	Full and Half Duplex Operation . . . . .	16-4
16.2.2	Interface Options . . . . .	16-4
16.2.2.1	10 Mbps and 100 Mbps MII Interface . . . . .	16-4
16.2.2.2	10 Mpbs 7-Wire Interface Operation . . . . .	16-4
16.2.3	Address Recognition Options . . . . .	16-4

16.2.4	Internal Loopback .....	16-4
16.3	External Signal Description .....	16-4
16.4	Memory Map/Register Definition .....	16-5
16.4.1	Ethernet Interrupt Event Register (EIR) .....	16-6
16.4.2	Interrupt Mask Register (EIMR) .....	16-8
16.4.3	Receive Descriptor Active Register (RDAR) .....	16-8
16.4.4	Transmit Descriptor Active Register (TDAR) .....	16-9
16.4.5	Ethernet Control Register (ECR) .....	16-10
16.4.6	MII Management Frame Register (MMFR) .....	16-10
16.4.7	MII Speed Control Register (MSCR) .....	16-12
16.4.8	Receive Control Register (RCR) .....	16-13
16.4.9	Transmit Control Register (TCR) .....	16-14
16.4.10	Physical Address Lower Register (PALR) .....	16-15
16.4.11	Physical Address Upper Register (PAUR) .....	16-15
16.4.12	Opcode/Pause Duration Register (OPD) .....	16-16
16.4.13	Descriptor Individual Upper Address Register (IAUR) .....	16-16
16.4.14	Descriptor Individual Lower Address Register (IALR) .....	16-17
16.4.15	Descriptor Group Upper Address Register (GAUR) .....	16-17
16.4.16	Descriptor Group Lower Address Register (GALR) .....	16-18
16.4.17	Transmit FIFO Watermark Register (TFWR) .....	16-18
16.4.18	FIFO Receive Bound Register (FRBR) .....	16-19
16.4.19	FIFO Receive Start Register (FRSR) .....	16-19
16.4.20	Receive Descriptor Ring Start Register (ERDSR) .....	16-20
16.4.21	Transmit Buffer Descriptor Ring Start Registers (ETSDR) .....	16-20
16.4.22	Receive Buffer Size Register (EMRBR) .....	16-21
16.5	Functional Description .....	16-21
16.5.1	Buffer Descriptors .....	16-22
16.5.1.1	Driver/DMA Operation with Buffer Descriptors .....	16-22
16.5.1.2	Ethernet Receive Buffer Descriptor (RxBD) .....	16-23
16.5.1.3	Ethernet Transmit Buffer Descriptor (TxBD) .....	16-25
16.5.2	Initialization Sequence .....	16-27
16.5.2.1	Hardware Controlled Initialization .....	16-27
16.5.3	User Initialization (Prior to Setting ECR[ETHER_EN]) .....	16-27
16.5.4	Microcontroller Initialization .....	16-28
16.5.5	User Initialization (After Setting ECR[ETHER_EN]) .....	16-28
16.5.6	Network Interface Options .....	16-28
16.5.7	FEC Frame Transmission .....	16-29
16.5.7.1	Duplicate Frame Transmission .....	16-30
16.5.8	FEC Frame Reception .....	16-31
16.5.9	Ethernet Address Recognition .....	16-31
16.5.10	Hash Algorithm .....	16-34
16.5.11	Full Duplex Flow Control .....	16-37
16.5.12	Inter-Packet Gap (IPG) Time .....	16-38
16.5.13	Collision Managing .....	16-38
16.5.14	MII Internal and External Loopback .....	16-38

16.5.15 Ethernet Error-Managing Procedure .....	16-38
16.5.15.1 Transmission Errors .....	16-39
16.5.15.2 Reception Errors .....	16-39

## Chapter 17

### Inter-Integrated Circuit (IIC)

17.1 Introduction .....	17-1
17.1.1 Features .....	17-2
17.1.2 Modes of Operation .....	17-2
17.1.3 Block Diagram .....	17-3
17.2 External Signal Description .....	17-3
17.2.1 SCL — Serial Clock Line .....	17-3
17.2.2 SDA — Serial Data Line .....	17-3
17.3 Register Definition .....	17-4
17.3.1 Module Memory Map .....	17-4
17.3.2 IIC Address Register 1 (IICA1) .....	17-4
17.3.3 IIC Frequency Divider Register (IICF) .....	17-5
17.3.4 IIC Control Register (IICC1) .....	17-8
17.3.5 IIC Status Register (IICS) .....	17-9
17.3.6 IIC Data I/O Register (IICD) .....	17-10
17.3.7 IIC Control Register 2 (IICC2) .....	17-12
17.3.8 IIC SMBus Control and Status Register (IICSMB) .....	17-13
17.3.9 IIC Address Register 2 (IICA2) .....	17-14
17.3.10 IIC SCL Low Time Out Register High (IICSLTH) .....	17-14
17.3.11 IIC SCL LowTime Out register Low (IICSLTL) .....	17-14
17.3.12 IIC Programmable Input Glitch Filter (IICFLT) .....	17-15
17.4 Functional Description .....	17-16
17.4.1 IIC Protocol .....	17-16
17.4.1.1 START Signal .....	17-17
17.4.1.2 Slave Address Transmission .....	17-17
17.4.1.3 Data Transfer .....	17-17
17.4.1.4 STOP Signal .....	17-18
17.4.1.5 Repeated START Signal .....	17-18
17.4.1.6 Arbitration Procedure .....	17-18
17.4.1.7 Clock Synchronization .....	17-18
17.4.1.8 Handshaking .....	17-19
17.4.1.9 Clock Stretching .....	17-19
17.4.2 10-bit Address .....	17-20
17.4.2.1 Master-Transmitter Addresses a Slave-Receiver .....	17-20
17.4.2.2 Master-Receiver Addresses a Slave-Transmitter .....	17-20
17.4.3 Address Matching .....	17-21
17.4.4 System Management Bus Specification .....	17-21
17.4.4.1 Timeouts .....	17-21
17.4.4.2 FAST ACK and NACK .....	17-23
17.5 Resets .....	17-23

17.6	Interrupts .....	17-23
17.6.1	Byte Transfer Interrupt .....	17-24
17.6.2	Address Detect Interrupt .....	17-24
17.6.3	Arbitration Lost Interrupt .....	17-24
17.6.4	Timeouts Interrupt in SMBus .....	17-24
17.6.5	Programmable input glitch filter .....	17-25
17.7	Initialization/Application Information .....	17-26

## Chapter 18 Modulo Timer (MTIM)

18.1	Introduction .....	18-1
18.1.1	Features .....	18-2
18.1.2	Modes of Operation .....	18-2
18.1.2.1	MTIM in Wait Mode .....	18-2
18.1.2.2	MTIM in Stop Modes .....	18-2
18.1.2.3	MTIM in Active Background Mode .....	18-2
18.1.3	Block Diagram .....	18-3
18.2	External Signal Description .....	18-3
18.3	Register Definition .....	18-4
18.3.1	MTIM Status and Control Register (MTIMSC) .....	18-5
18.3.2	MTIM Clock Configuration Register (MTIMCLK) .....	18-6
18.3.3	MTIM Counter Register (MTIMCNT) .....	18-7
18.3.4	MTIM Modulo Register (MTIMMOD) .....	18-7
18.4	Functional Description .....	18-8
18.4.1	MTIM Operation Example .....	18-9

## Chapter 19 Timer/PWM Module (TPM)

19.1	Introduction .....	19-1
19.1.1	Features .....	19-2
19.1.2	Modes of Operation .....	19-2
19.1.3	Block Diagram .....	19-3
19.2	Signal Description .....	19-5
19.2.1	Detailed Signal Descriptions .....	19-5
19.2.1.1	EXTCLK — External Clock Source .....	19-5
19.2.1.2	TPMxCHn — TPM Channel n I/O Pins .....	19-5
19.3	Register Definition .....	19-8
19.3.1	TPM Status and Control Register (TPMxSC) .....	19-8
19.3.2	TPM-Counter Registers (TPMxCNTH:TPMxCNTL) .....	19-9
19.3.3	TPM Counter Modulo Registers (TPMxMODH:TPMxMODL) .....	19-10
19.3.4	TPM Channel n Status and Control Register (TPMxCnSC) .....	19-11
19.3.5	TPM Channel Value Registers (TPMxCnVH:TPMxCnVL) .....	19-12
19.4	Functional Description .....	19-13
19.4.1	Counter .....	19-14

19.4.1.1	Counter Clock Source .....	19-14
19.4.1.2	Counter Overflow and Modulo Reset .....	19-14
19.4.1.3	Counting Modes .....	19-15
19.4.1.4	Manual Counter Reset .....	19-15
19.4.2	Channel Mode Selection .....	19-15
19.4.2.1	Input Capture Mode .....	19-15
19.4.2.2	Output Compare Mode .....	19-15
19.4.2.3	Edge-Aligned PWM Mode .....	19-16
19.4.2.4	Center-Aligned PWM Mode .....	19-17
19.5	Reset Overview .....	19-18
19.5.1	General .....	19-18
19.5.2	Description of Reset Operation .....	19-18
19.6	Interrupts .....	19-18
19.6.1	General .....	19-18
19.6.2	Description of Interrupt Operation .....	19-19
19.6.2.1	Timer Overflow Interrupt (TOF) Description .....	19-19
19.6.2.2	Channel Event Interrupt Description .....	19-19

## Chapter 20

### Version 1 ColdFire Debug (CF1\_DEBUG)

20.1	Introduction .....	20-1
20.1.1	Overview .....	20-2
20.1.2	Features .....	20-3
20.1.3	Modes of Operations .....	20-3
20.2	External Signal Descriptions .....	20-5
20.3	Memory Map/Register Definition .....	20-6
20.3.1	Configuration/Status Register (CSR) .....	20-7
20.3.2	Extended Configuration/Status Register (XCSR) .....	20-10
20.3.3	Configuration/Status Register 2 (CSR2) .....	20-13
20.3.4	Configuration/Status Register 3 (CSR3) .....	20-16
20.3.5	BDM Address Attribute Register (BAAR) .....	20-17
20.3.6	Address Attribute Trigger Register (AATR) .....	20-18
20.3.7	Trigger Definition Register (TDR) .....	20-19
20.3.8	Program Counter Breakpoint/Mask Registers (PBR0–3, PBMR) .....	20-22
20.3.9	Address Breakpoint Registers (ABLR, ABHR) .....	20-24
20.3.10	Data Breakpoint and Mask Registers (DBR, DBMR) .....	20-25
20.3.10.1	Resulting Set of Possible Trigger Combinations .....	20-26
20.3.11	PST Buffer (PSTB) .....	20-26
20.4	Functional Description .....	20-27
20.4.1	Background Debug Mode (BDM) .....	20-27
20.4.1.1	CPU Halt .....	20-28
20.4.1.2	Background Debug Serial Interface Controller (BDC) .....	20-29
20.4.1.3	BDM Communication Details .....	20-30
20.4.1.4	BDM Command Set Descriptions .....	20-33
20.4.1.5	BDM Command Set Summary .....	20-36

20.4.1.6	Serial Interface Hardware Handshake Protocol . . . . .	20-51
20.4.1.7	Hardware Handshake Abort Procedure . . . . .	20-53
20.4.2	Real-Time Debug Support . . . . .	20-56
20.4.3	Trace Support . . . . .	20-56
20.4.3.1	Begin Execution of Taken Branch (PST = 0x05) . . . . .	20-58
20.4.3.2	PST Trace Buffer (PSTB) Entry Format . . . . .	20-59
20.4.3.3	PST/DDATA Example . . . . .	20-59
20.4.3.4	Processor Status, Debug Data Definition . . . . .	20-61
20.4.4	Freescale-Recommended BDM Pinout . . . . .	20-66

## About This Book

The primary objective of this reference manual is to define the MCF51CN128 processor for software and hardware developers. This book is written from the perspective of the MCF51CN128. The information in this book is subject to change without notice, as described in the disclaimers on the title page. As with any technical documentation, the reader needs to make sure to use the most recent version of the documentation.

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

## Audience

This manual is intended for system software and hardware developers and applications programmers who want to develop products with this ColdFire processor. 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.

## Suggested Reading

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

## General Information

Useful information about the ColdFire architecture and computer architecture in general:

- *ColdFire Programmers Reference Manual* (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, as well as our web site, <http://www.freescale.com/coldfire>.

- Reference manuals — These books provide details about individual ColdFire implementations and are intended to be used in conjunction with the *ColdFire Programmers Reference Manual*.
- Data sheets — Data sheets provide specific data regarding pin-out diagrams, bus timing, signal behavior, and AC, DC, and thermal characteristics, as well as other design considerations.

- Product briefs — Each device has a product brief that provides an overview of its features. This document is roughly equivalent to the overview (Chapter 1) of an device’s reference manual.
- Application notes — These short documents address specific design issues useful to programmers and engineers working with Freescale Semiconductor processors.

Additional literature is published as new processors become available. For a current list of ColdFire documentation, refer to <http://www.freescale.com/coldfire>.

## Conventions

This document uses the following notational conventions:

cleared/set	When a bit takes the value zero, it is said to be cleared; when it takes a value of one, it is said to be set.
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
x	In some contexts, such as signal encodings, x indicates a don’t care.
n	Used to express an undefined numerical value
~	NOT logical operator
&	AND logical operator
	OR logical operator
	Field concatenation operator
OVERBAR	An overbar indicates that a signal is active-low.

## Register Figure Conventions

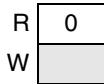
This document uses the following conventions for the register reset values:

—	Undefined at reset.
u	Unaffected by reset.

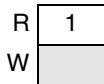
<sup>1</sup>The only exceptions to this appear in the discussion of serial communication modules that support variable-length data transmission units. To simplify the discussion these units are referred to as words regardless of length.

[*signal\_name*] Reset value is determined by the polarity of the indicated signal.

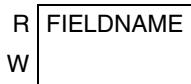
The following register fields are used:



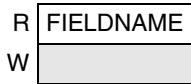
Indicates a reserved bit field in a memory-mapped register. These bits are always read as zeros.



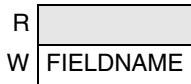
Indicates a reserved bit field in a memory-mapped register. These bits are always read as ones.



Indicates a read/write bit.



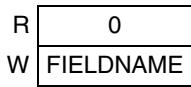
Indicates a read-only bit field in a memory-mapped register.



Indicates a write-only bit field in a memory-mapped register.



Write 1 to clear: indicates that writing a 1 to this bit field clears it.



Indicates a self-clearing bit.

# Chapter 1

## Device Overview

### 1.1 The MCF51CN128 Series Microcontrollers

#### 1.1.1 Definition

The MCF51CN128 series microcontrollers are systems-on-chips (SoCs) that are based on the V1 ColdFire core and:

- Operate at processor core speeds up to 50.33 MHz (all peripherals except the SCIs operate at half of this speed. SCIs are clocked at the CPU rate.)
- Feature 10/100 BASE-T/TX fast ethernet controller (FEC) with media independent interface (MII) to connect an external physical transceiver (PHY)
- Multi-function external bus interface (Mini-FlexBus).
- A collection of communications peripherals, including UART, IIC and SPI.
- Integrated 12-bit SAR Analog-to-Digital Converter

These devices are ideal for low-cost ethernet-enabled applications, including smart-sensor applications.

#### 1.1.2 MCF51CN128 Series Devices

The MCF51CN128 devices are available in various packages, as shown in [Table 1-1](#).

**Table 1-1. MCF51CN128 Series Package Availability**

	MCF51CN128
80-pin LQFP	Yes
64-pin LQFP	Yes
48-pin QFN	Yes

### 1.1.3 MCF51CN128 Series Device Comparison

Table 1-2 compares the MCF51CN128 series microcontrollers.

**Table 1-2. MCF51CN128 Series Device Comparison**

Feature	MCF51CN128		
	80-pin	64-pin	48-pin
Flash memory size (KB)	128		
RAM size (KB)	24		
V1 ColdFire core equipped with BDM (background debug module) and 2X3 Crossbar switch	Yes		
ADC (analog-to-digital converter) channels (12-bit)	12		
FEC (Fast Ethernet Controller with MII Interface)	Yes		
COP (computer operating properly)	Yes		
IIC1 (inter-integrated circuit)	Yes		
IIC2	Yes		
IRQ (interrupt request input)	Yes		
KBI (keyboard interrupts)	16	12	6
LVD (low-voltage detector)	Yes		
MCG (multipurpose clock generator)	Yes		
Port I/O <sup>1</sup>	70	54	38
GPIO (rapid general-purpose I/O)	16	16	8
RTC (real-time counter)	Yes		
SCI1, SCI2 & SCI3 (serial communications interface)	Yes		
SPI1 & SPI2 (serial peripheral interface)	Yes		
TPM1 (Timer/PWM Module) channels	3	3	3
TPM2 channels	3	3	3
MTIM1 & MTIM2	Yes <sup>2</sup>		
External Timer Clocks	2	1	1
Mini-FlexBus	Yes	0	0
XOSC (crystal oscillator)	Yes		

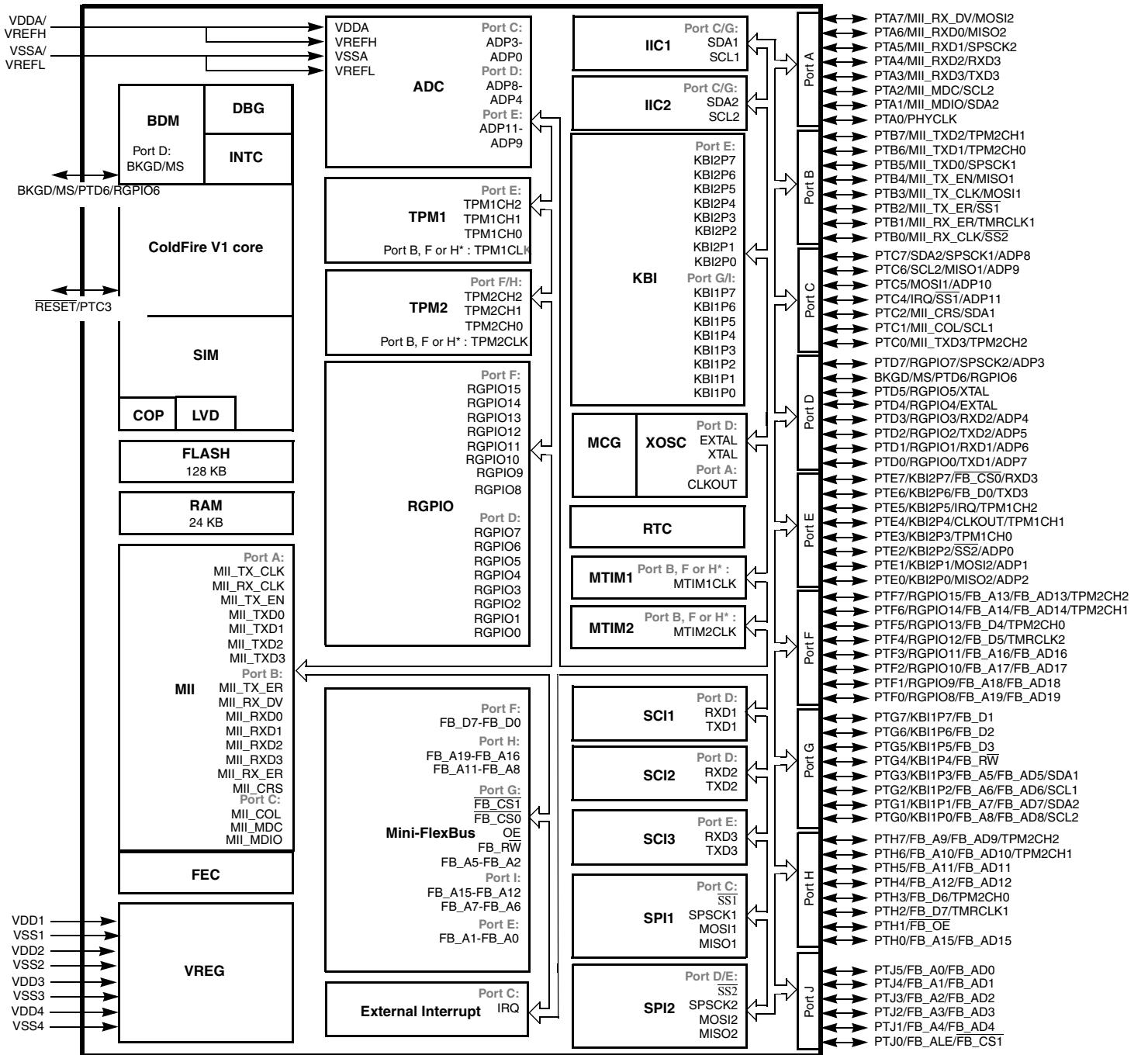
<sup>1</sup> All GPIO are muxed with other functions

<sup>2</sup> TMRCLK2 is not available on the 48 pin package, although MTIM2 can be used as an internal timebase using on-chip clock sources.

## 1.2 MCF51CN128 Series Block Diagram

### 1.2.1 Block Diagram

Figure 1-1 shows the connections between the MCF51CN128 series pins and functional units.



\* TPMx and MTIMx external clocks each have the choice of being assigned to either TMRCLK1 or TMRCLK2. See Section 5.7.14, "SIM Internal Peripheral Select Register (SIMIPS)", for details.

Figure 1-1. MCF51CN128 Series Block Diagram

## 1.2.2 Functional Units

Table 1-3 describes the functional units of the MCF51CN128 series microcontrollers.

**Table 1-3. MCF51CN128 Series Functional Units**

Unit	Function
ADC (analog-to-digital converter)	Measures analog voltages at up to 12 bits of resolution
BDM (background debug module)	Provides single pin debugging interface (part of the V1 ColdFire core)
CF1CORE (V1 ColdFire core)	Executes programs and interrupt handlers.
COP (Computer Operating Properly)	Software Watchdog
DBG (debug)	Provides debugging and emulation capabilities (part of the V1 ColdFire core)
FEC (fast ethernet controller) with IEEE-compliant MII (media independent interface)	10/100 BASE-T/TX, bus-mastering fast ethernet controller with direct memory access (DMA); supports half or full duplex
FLASH (flash memory)	Provides storage for program code, constants and variables
IIC1, IIC2 (inter-integrated circuits)	Supports standard IIC communications protocol and SMBus
INTC (interrupt controller)	Controls and prioritizes all device interrupts
KBI (keyboard interrupt)	Provides pin interrupt capabilities
LVD (low voltage detect)	Provides an interrupt to the CF1CORE in the event that the supply voltage drops below a critical value. The LVD can also be programmed to reset the device upon a low voltage event.
MCG (multipurpose clock generator)	Provides clocking options for the device, including a phase-locked loop(PLL) and frequency-locked loop (FLL) for multiplying slower reference clock sources
Mini-FlexBus	Provides expansion capability for off-chip memory and peripherals
MTIM1, MTIM2 (modulo timers)	8-bit Modulo Timers with configurable clock inputs and interrupt generation on overflow.
RAM (random-access memory)	Provides stack and variable storage
GPIO (rapid general-purpose input/output)	Allows for I/O port access at CPU clock speeds
RTC (real-time counter)	Provides a constant time-base with optional interrupt
SCI1, SCI2, SCI3(serial communications interfaces)	Serial communications UARTs capable of supporting RS-232 and LIN protocols
SIM (System Integration Unit)	
SPI1, SPI2 (serial peripheral interfaces)	Provide 4-pin synchronous serial interface
TPM1, TPM2 (Timer/PWM Module)	Timer/PWM module can be used for a variety of generic timer operations as well as pulse-width modulation
VREG (voltage regulator)	Controls power management across the device
XOSC (crystal oscillator)	Supports low- or high-range crystals.

### 1.2.3 Module Versions

Table 1-4 provides the functional version of the on-chip modules

**Table 1-4. Module Versions**

Module	Version
Analog-to-Digital Converter (ADC12)	1
Fast Ethernet Controller (FEC)	
General Purpose I/O (GPIO)	2
Inter-Integrated Circuit (IIC)	3
Interrupt Controller (CF1_INTC)	1
Keyboard Interrupt (KBI)	2
Low Power Oscillator (OSCVLP_25MHz )	1
Mini-FlexBus	
Modulo Timer (MTIM)	
Multi-purpose Clock Generator (MCG)	3
Real-Time Counter (RTC)	1
Serial Communications Interface (SCI)	4
Serial Peripheral Interface (SPI)	3
Timer/PWM Module (TPM)	3
V1 ColdFire Core (CF1CORE)	1
Voltage Regulator (PMCx)	1

## 1.3 V1 ColdFire Core

The MCF51CN128 series devices contain a version of the V1 ColdFire platform that is optimized for area and low power. The CPU implements ColdFire instruction set architecture revision C (ISA\_C) with a reduced programming model:

- No hardware support for MAC/EMAC and DIV instructions
- Upward compatibility with other ColdFire cores (V2–V5)

An integrated multi-master crossbar switch on the ColdFire system busses provides access to system resources by the CPU and Ethernet controller. The CPU configuration register has one bit for control of the crossbar switch.

For more details on the V1 ColdFire core, see [Chapter 7, “ColdFire Core.”](#)

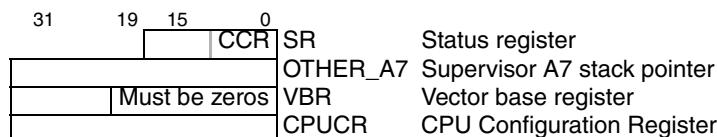
### 1.3.1 User Programming Model

[Figure 1-2](#) illustrates the integer portion of the user programming model. It consists 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)

### 1.3.2 Supervisor Programming Model

System programmers use the supervisor programming model to implement operating system functions. All accesses that affect the control features of ColdFire processors must be made in supervisor mode and can be accessed only by privileged instructions. The supervisor programming model consists of the registers available in user mode and the registers listed in [Figure 1-2](#).

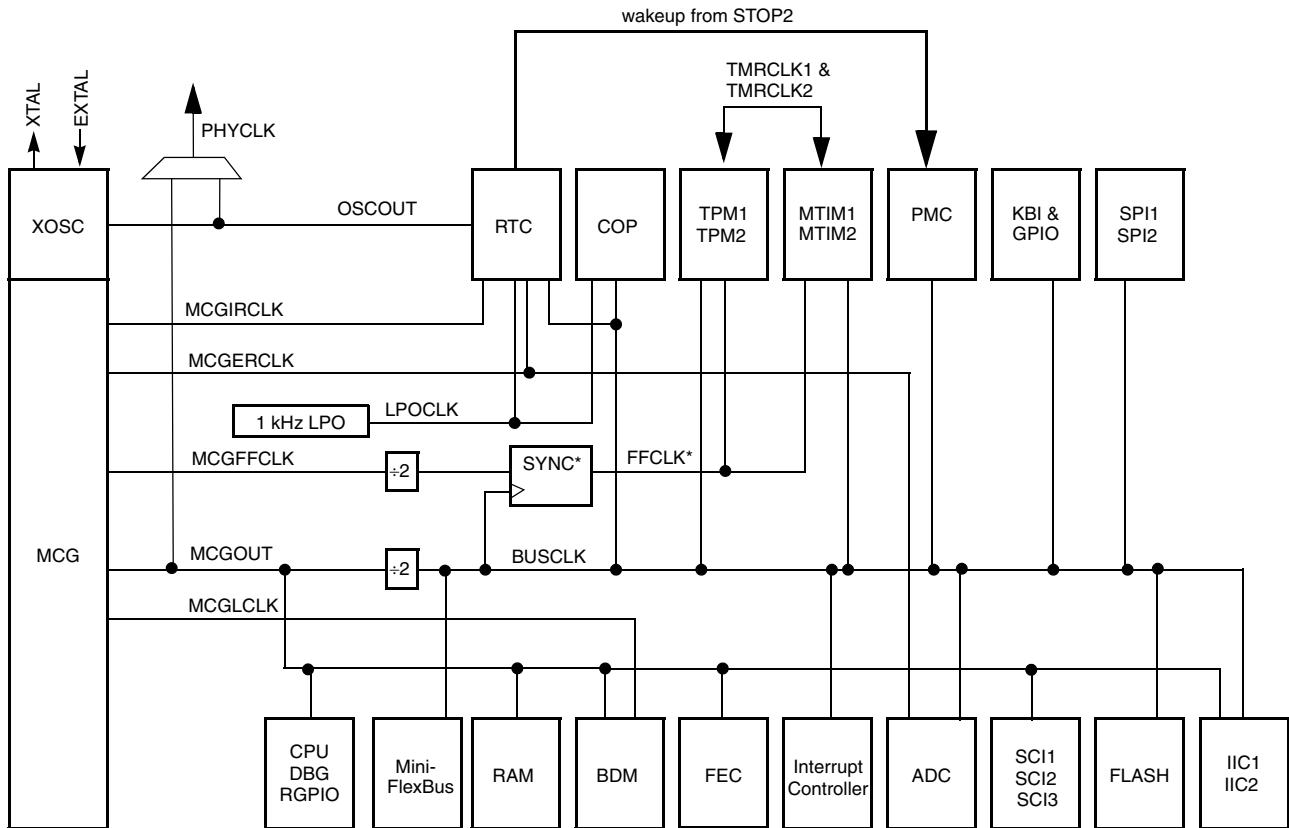


**Figure 1-2. Supervisor Programming Model**

## 1.4 System Clock Generation and Distribution

### 1.4.1 Clock Distribution Diagram

Figure 1-3 shows how clocks from the MCG and XOSC are distributed to the microcontroller's other functional units. Some modules in the microcontroller have selectable clock inputs. All memory-mapped registers associated with the modules (except GPIO) are clocked with BUSCLK. The GPIO registers are clocked with the CPU clock (MCGOUT).



#### Notes

- <sup>1</sup> The ADC has minimum and maximum frequency requirements. See the [Chapter 15, “Analog-to-Digital Converter \(ADC12\)”](#) and the [MCF51CN128 Data Sheet](#).
- <sup>2</sup> Flash memory has frequency requirements for program and erase operations. See the [MCF51CN128 Data Sheet](#).
- <sup>3</sup> IICx module uses the full speed core clock for input filtering only.
- <sup>4</sup> \* The fixed frequency clock (FFCLK) is internally synchronized to the bus clock (BUSCLK) and must not exceed one half of the bus clock frequency.

**Figure 1-3. Clock Distribution Diagram**

The TPMx and MTIMx external clocks can be assigned to TMRCLK1 or TMRCLK2. See [Section 5.7.14, “SIM Internal Peripheral Select Register \(SIMIPS\)”](#), for details

## 1.4.2 System Clocks

Table 1-5 describes each of the system clocks.

**Table 1-5. System Clocks**

Clock	Description
OSCOUT	This is the direct output of the external oscillator module and can be selected as the real-time counter clock source. This signal is used by the Real Time Counter, and the MCG. See <a href="#">Chapter 6, “Multipurpose Clock Generator (MCG)”</a> and <a href="#">Chapter 12, “Real-Time Counter (RTC),”</a> for details.
MCGOUT	This clock drives the CPU, debug, RAM, and BDM directly and is divided by two to clock all peripherals (BUSCLK). Control bits in the MCG control registers determine which of three clock sources is connected: <ul style="list-style-type: none"> <li>• Internal reference clock</li> <li>• External reference clock</li> <li>• Frequency-locked loop (FLL) or phase-locked loop (PLL) output</li> </ul> See <a href="#">Chapter 6, “Multipurpose Clock Generator (MCG),”</a> for details on configuring the MCGOUT clock.
MCGLCLK	This clock source runs at half the rate of the low frequency DCO (digitally controlled oscillator). Development tools can select this internal self-coded source to speed up BDC communications in systems where the bus clock is slow. Please see <a href="#">Figure 6-1</a> , <a href="#">Figure 20-2</a> , and their accompanying text for details.
MGERCLK	This is the external reference clock and can be selected as an alternate clock for the ADC.
MGIRCLK	This is the internal reference clock and can be selected as the real-time counter clock source.
MGFFCLK	This is the fixed frequency clock (FFCLK) after being synchronized to the bus clock. It can be selected as a clock source for the TPM modules. The frequency of the FFCLK is determined by the settings of the MCG.
LPOCLK	This clock is generated from an internal low-power oscillator that is completely independent of the MCG module. The LPOCLK can be selected as the clock source to the RTC or COP.
TMRCLK1 & TMRCLK2	Optional external clock sources for the TPMs and MTIMs. These clocks must be limited to one-quarter the frequency of the bus clock for synchronization.
ADACK (not shown)	The ADC module also has an internally generated asynchronous clock which allows it to run in STOP mode. This signal is not available externally.
CLKOUT	This is an optional output of the device which can be used to deliver the crystal oscillator output or the output of the PLL off-chip.

## 1.4.3 Clock Gating

To save power, peripheral clocks can be shut off by programming the system clock gating registers. For details, refer to [Section 5.7.10, “System Clock Gating Control 1 Register \(SCGC1\).”](#)

## 1.4.4 MCG Modes of Operation

The MCG operates in one of the modes described in [Table 1-6](#). This information has been abbreviated for clarity's sake. See the MCG block specification for additional details.

**Table 1-6. MCG Modes**

Mode	Description
FLL Engaged Internal (FEI)	Default. MCGOUT is derived from the FLL clock, which is controlled by the internal reference clock. The FLL clock frequency locks to a multiple of the internal reference frequency. MCGLCLK is derived from the FLL, and the PLL is disabled in a low-power state.
FLL Engaged External (FEE)	MCGOUT is derived from the FLL clock, which is controlled by the external reference clock. The external reference clock that is enabled can be produced by an external crystal, ceramic resonator, or another external clock source connected to the required crystal oscillator (XOSC).
FLL Bypassed Internal (FBI)	MCGOUT is derived from the internal reference clock; the FLL is operational, but its output clock is not used. This mode is useful to allow the FLL to acquire its target frequency while the MCGOUT clock is driven from the internal reference clock.
FLL Bypassed External (FBE)	MCGOUT is derived from the external reference clock; the FLL is operational, but its output clock is not used. This mode is useful to allow the FLL to acquire its target frequency while MCGOUT is driven from the external reference clock.
PLL Engaged External (PEE)	MCGOUT is derived from the PLL clock, which is controlled by the external reference clock. The external reference clock that is enabled can be produced by an external crystal, ceramic resonator, or another external clock source connected to the required crystal oscillator (XOSC).
PLL Bypassed External (PBE)	MCGOUT is derived from the external reference clock; the PLL is operational, but its output clock is not used by the CPU. The external reference clock that is enabled can be produced by an external crystal, ceramic resonator, or another external clock source connected to the required crystal oscillator (XOSC). This mode is useful to allow the PLL to acquire its target frequency while MCGOUT is driven from the external reference clock. This mode could also be used to source the Ethernet MAC clock while running the CPU at a lower clock rate.
Bypassed Low Power Internal (BLPI)	MCGOUT is derived from the internal reference clock. The PLL and FLL are disabled, and MCGLCLK is not available for BDC communications. If the BDM becomes enabled, the mode switches to one of the bypassed internal modes.

**Table 1-6. MCG Modes (continued)**

<b>Mode</b>	<b>Description</b>
Bypassed Low Power External (BLPE)	MCGOUT is derived from the external reference clock. The external reference clock that is enabled can be produced by an external crystal, ceramic resonator, or another external clock source connected to the required crystal oscillator (XOSC). The PLL and FLL are disabled, and MCGLCLK is not available for BDC communications. If the BDM becomes enabled, the mode switches to one of the bypassed external mode.
STOP	Entered whenever the MCU enters a Stop state. The FLL and PLL are disabled, and all MCG clock signals are static except in the following cases: <ul style="list-style-type: none"> <li>• MCGIRCLK can be active in Stop mode under certain conditions.</li> </ul>

### 1.4.5 MCG Mode State Diagram

Figure 1-4 shows the valid state transitions for the MCG. The arrows indicate the permitted mode transitions. See Chapter 6, “Multipurpose Clock Generator (MCG),” for additional details.

Software must ensure that the system bus frequency is less than 125 kHz and the FLLs are disengaged prior to switching to BLPE and BLPI modes of operation.

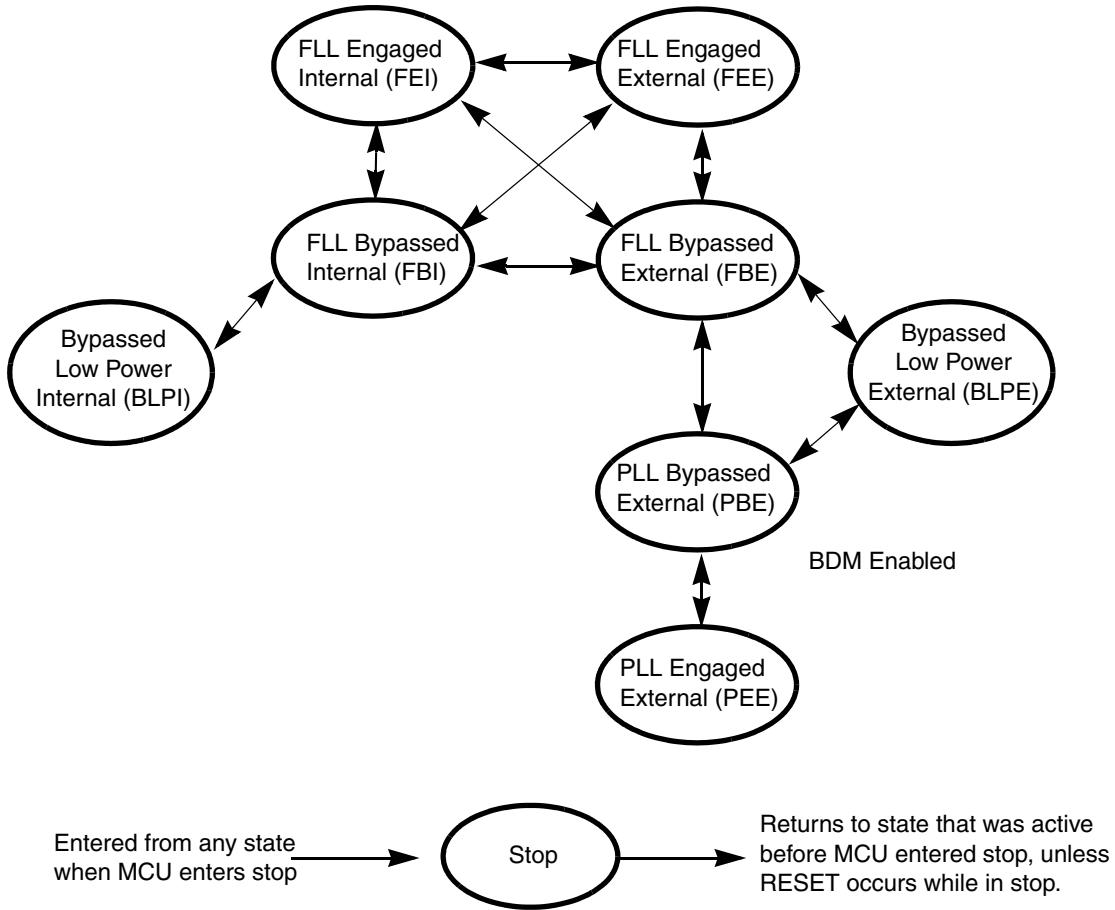


Figure 1-4. MCG Mode State Diagram

# **Chapter 2**

## **Pins and Connections**

This section describes signals that connect to package pins. It includes pinout diagrams, recommended system connections, and detailed discussions of signals.

### **2.1 Package Pin Assignments**

#### **2.1.1 Pinout: 80-Pin LQFP**

[Figure 2-1](#) shows the pinout of the 80-pin LQFP.

## Pins and Connections

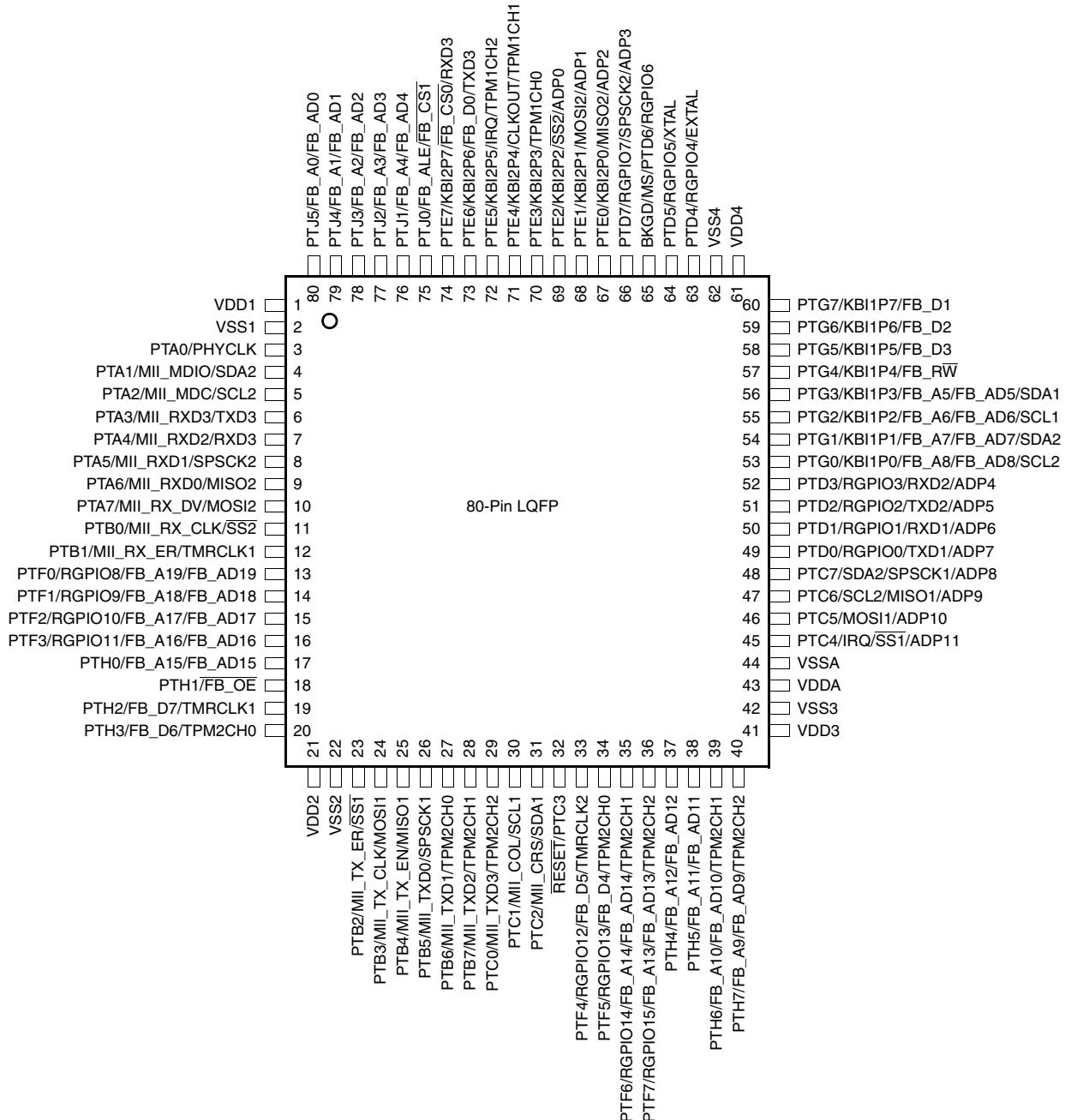


Figure 2-1. 80-Pin LQFP

## 2.1.2 Pinout: 64-Pin LQFP

Figure 2-2 shows the pinout of the 64-pin LQFP.

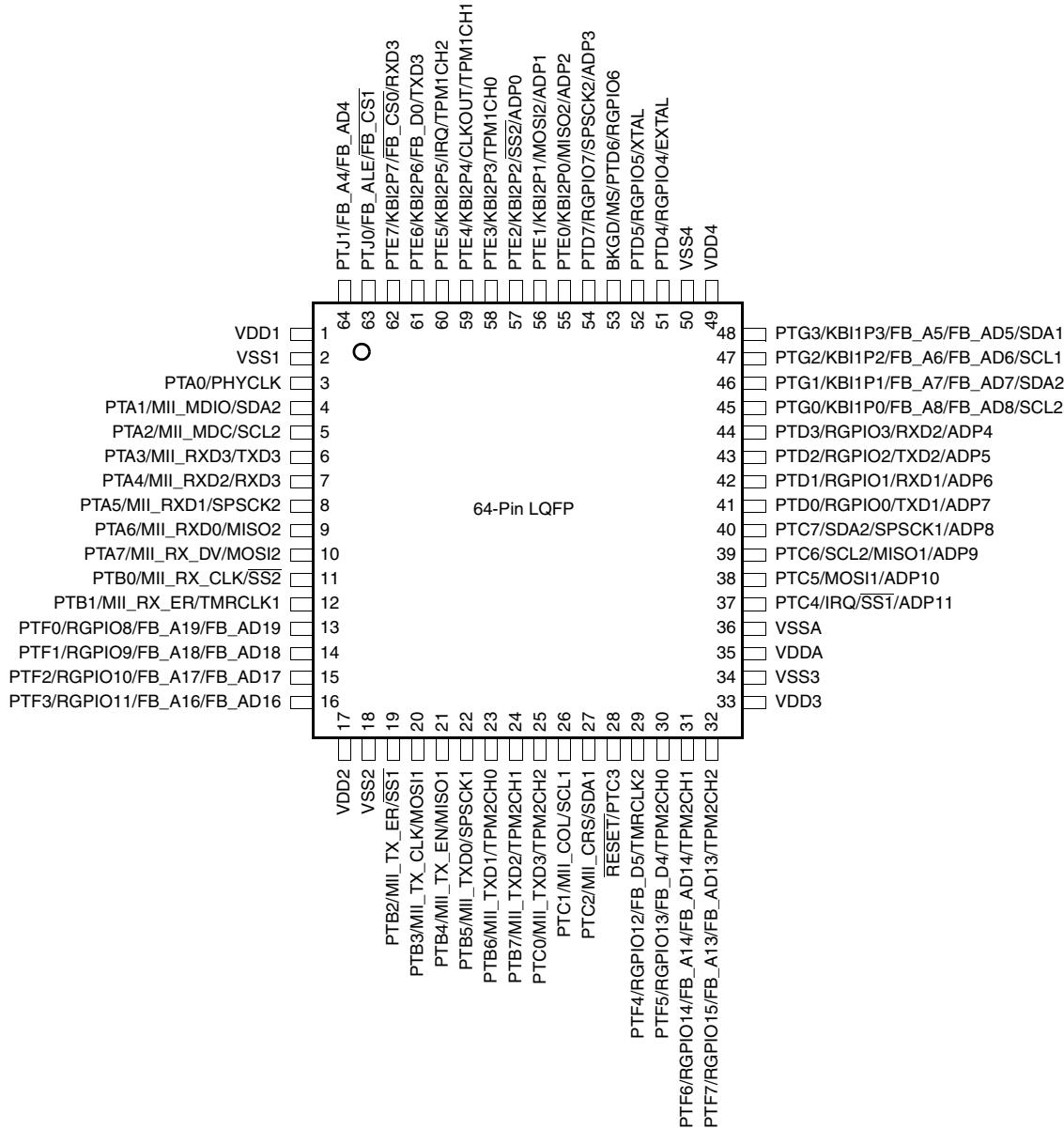


Figure 2-2. 64-Pin LQFP

### 2.1.3 Pinout: 48-Pin QFN

Figure 2-3 shows the pinout of the 48-pin QFN.

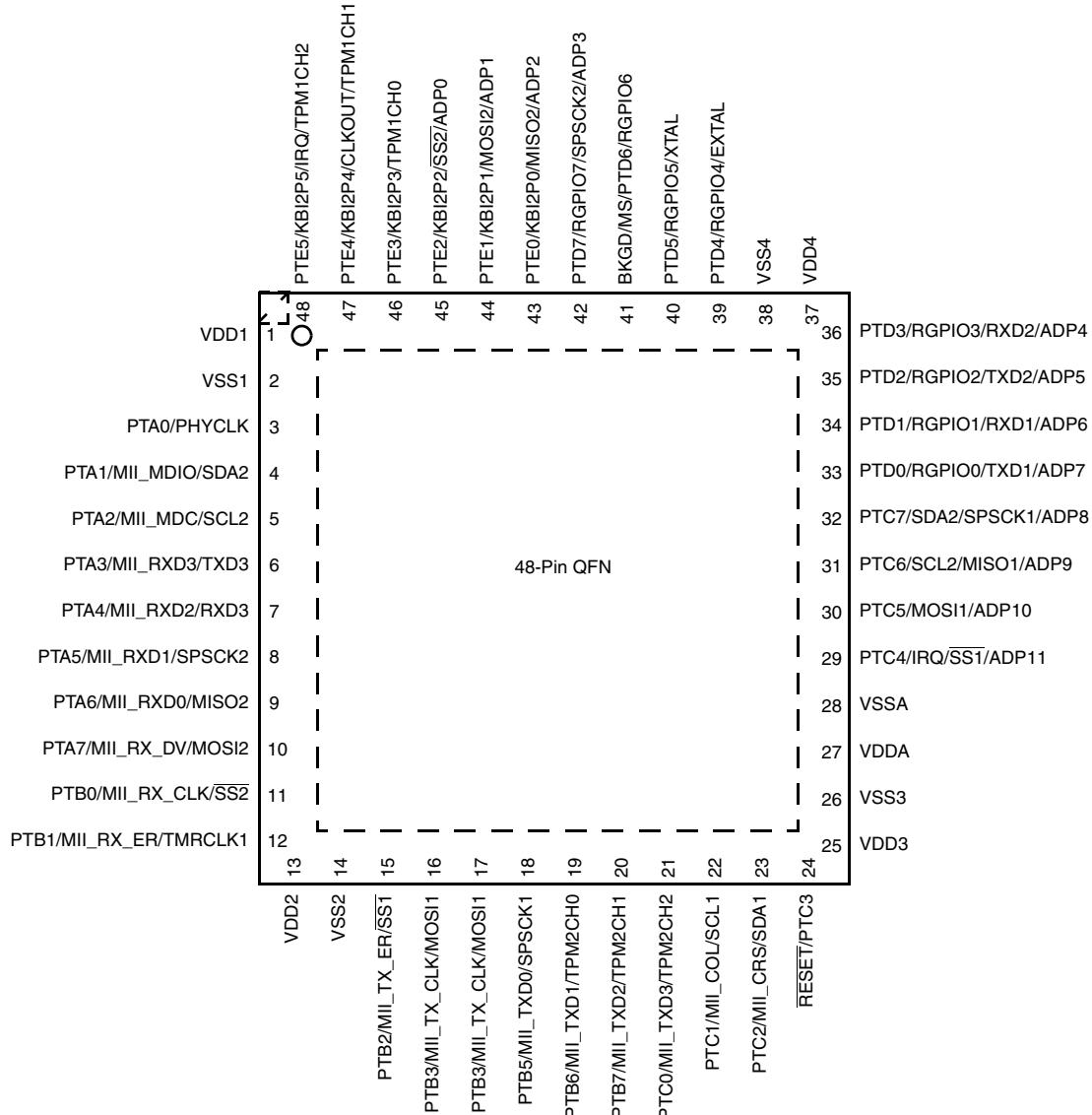


Figure 2-3. 48-Pin QFN

#### NOTE

There is no electrical connection to the flag for 48-pin QFN packages.

## 2.2 Pin Assignment Tables

MCF51CN128 family is available in 80-pin LQFP, 64-pin LQFP, and 48-pin QFN package options. Table 2-1 summarizes the functions available on each pin of the various package configurations. The default function column specifies the function of the given pin upon exiting the reset state. Alternate functions 1, 2 and 3 can be assigned to each pin under software control via the MC<sup>1</sup> registers.

For proper operation, a given peripheral function must be assigned to a maximum of one package pin. Do not program the same function to two or more pins.

Peripherals which are not available in smaller packages have their input pins and peripheral clocks disabled by default. The mux controls for keyboard 1 and the Mini-FlexBus must not be programmed to those functions unless supported on the package in question.

**Table 2-1. MC51CN128 Package Pin Assignments**

80-Pin	64-Pin	48-Pin	Default Function	Alt 1	Alt 2	Alt 3	Comment
1	1	1	VDD1	—	—	—	—
2	2	2	VSS1	—	—	—	—
3	3	3	PTA0	PHYCLK	—	—	—
4	4	4	PTA1	MII_MDIO	—	SDA2	—
5	5	5	PTA2	MII_MDC	—	SCL2	—
6	6	6	PTA3	MII_RXD3	TXD3	—	—
7	7	7	PTA4	MII_RXD2	RXD3	—	—
8	8	8	PTA5	MII_RXD1	SPSCK2	—	—
9	9	9	PTA6	MII_RXD0	MISO2	—	—
10	10	10	PTA7	MII_RX_DV	MOSI2	—	—
11	11	11	PTB0	MII_RX_CLK	SS2	—	—
12	12	12	PTB1	MII_RX_ER	—	TMRCLK1	—
13	13	—	PTF0/RGPIO8	—	FB_A19/FB_AD19	—	RGPIO_ENB selects between standard GPIO and RGPIO
14	14	—	PTF1/RGPIO9	—	FB_A18/FB_AD18	—	
15	15	—	PTF2/RGPIO10	—	FB_A17/FB_AD17	—	
16	16	—	PTF3/RGPIO11	—	FB_A16/FB_AD16	—	
17	—	—	PTH0	—	FB_A15/FB_AD15	—	—
18	—	—	PTH1	—	FB_OE	—	—
19	—	—	PTH2	—	FB_D7	TMRCLK1	—
20	—	—	PTH3	—	FB_D6	TPM2CH0	—
21	17	13	VDD2	—	—	—	—
22	18	14	VSS2	—	—	—	—

1. Port Mux Control registers.

**Table 2-1. MC51CN128 Package Pin Assignments (continued)**

<b>80-Pin</b>	<b>64-Pin</b>	<b>48-Pin</b>	<b>Default Function</b>	<b>Alt 1</b>	<b>Alt 2</b>	<b>Alt 3</b>	<b>Comment</b>
23	19	15	PTB2	MII_TX_ER	SS1	—	—
24	20	16	PTB3	MII_TX_CLK	MOSI1	—	—
25	21	17	PTB4	MII_TX_EN	MISO1	—	—
26	22	18	PTB5	MII_TXD0	SPSCK1	—	—
27	23	19	PTB6	MII_TXD1	—	TPM2CH0	—
28	24	20	PTB7	MII_TXD2	—	TPM2CH1	—
29	25	21	PTC0	MII_TXD3	—	TPM2CH2	—
30	26	22	PTC1	MII_COL	—	SCL1	—
31	27	23	PTC2	MII_CRS	—	SDA1	—
32	28	24	RESET	PTC3	—	—	This pin is a bidirectional open drain device and has an internal pullup. There is no clamp diode to VDD. DSE and SRE port controls for this bit have no effect.
33	29	—	PTF4/RGPIO12	—	FB_D5	TMRCLK2	RGPIO_ENB selects between standard GPIO and RGPIO
34	30	—	PTF5/RGPIO13	—	FB_D4	TPM2CH0	
35	31	—	PTF6/RGPIO14	—	FB_A14/FB_AD14	TPM2CH1	
36	32	—	PTF7/RGPIO15	—	FB_A13/FB_AD13	TPM2CH2	
37	—	—	PTH4	—	FB_A12/FB_AD12	—	
38	—	—	PTH5	—	FB_A11/FB_AD11	—	—
39	—	—	PTH6	—	FB_A10/FB_AD10	TPM2CH1	—
40	—	—	PTH7	—	FB_A9/FB_AD9	TPM2CH2	—
41	33	25	VDD3	—	—	—	—
42	34	26	VSS3	—	—	—	—
43	35	27	VDDA	—	—	—	—
44	36	28	VSSA	—	—	—	—
45	37	29	PTC4	IRQ	SS1	ADP11	—
46	38	30	PTC5	—	MOSI1	ADP10	—
47	39	31	PTC6	SCL2	MISO1	ADP9	—
48	40	32	PTC7	SDA2	SPSCK1	ADP8	—

**Table 2-1. MC51CN128 Package Pin Assignments (continued)**

<b>80-Pin</b>	<b>64-Pin</b>	<b>48-Pin</b>	<b>Default Function</b>	<b>Alt 1</b>	<b>Alt 2</b>	<b>Alt 3</b>	<b>Comment</b>
49	41	33	PTD0/RGPIO0	—	TXD1	ADP7	RGPIO_ENB selects between standard GPIO and RGPIO
50	42	34	PTD1/RGPIO1	—	RXD1	ADP6	
51	43	35	PTD2/RGPIO2	—	TXD2	ADP5	
52	44	36	PTD3/RGPIO3	—	RXD2	ADP4	
53	45	—	PTG0	KBI1P0	FB_A8/FB_AD8	SCL2	
54	46	—	PTG1	KBI1P1	FB_A7/FB_AD7	SDA2	
55	47	—	PTG2	KBI1P2	FB_A6/FB_AD6	SCL1	—
56	48	—	PTG3	KBI1P3	FB_A5/FB_AD5	SDA1	—
57	—	—	PTG4	KBI1P4	FB_RW	—	—
58	—	—	PTG5	KBI1P5	FB_D3	—	—
59	—	—	PTG6	KBI1P6	FB_D2	—	—
60	—	—	PTG7	KBI1P7	FB_D1	—	—
61	49	37	VDD4	—	—	—	—
62	50	38	VSS4	—	—	—	—
63	51	39	PTD4/RGPIO4	—	—	EXTAL	RGPIO_ENB selects between standard GPIO and RGPIO
64	52	40	PTD5/RGPIO5	—	—	XTAL	
65	53	41	BKGD/MS	PTD6/RGPIO6	—	—	This pin has an internal pullup. PTD6/RGPIO6 can only be programmed as an output. RGPIO_ENB selects between standard GPIO and RGPIO. When PTD6 is set as RGPIO output, and "1" is driven to PTD6 via RGPIO function, a read of register RGPIODATA6 always returns a "0" because V1 RGPIO design looks for IO enable when the return value of RGPIO function reads data. As PTD6 is set to RGPIO output only, it returns "0" always to RGPIODATA6, although PTD6 pin is driven to high.

**Table 2-1. MC51CN128 Package Pin Assignments (continued)**

<b>80-Pin</b>	<b>64-Pin</b>	<b>48-Pin</b>	<b>Default Function</b>	<b>Alt 1</b>	<b>Alt 2</b>	<b>Alt 3</b>	<b>Comment</b>
66	54	42	PTD7RGPIO7	—	SPSCK2	ADP3	RGPIO_ENB selects between standard GPIO and RGPIO
67	55	43	PTE0	KBI2P0	MISO2	ADP2	—
68	56	44	PTE1	KBI2P1	MOSI2	ADP1	—
69	57	45	PTE2	KBI2P2	$\overline{SS2}$	ADP0	—
70	58	46	PTE3	KBI2P3	—	TPM1CH0	—
71	59	47	PTE4	KBI2P4	CLKOUT	TPM1CH1	—
72	60	48	PTE5	KBI2P5	IRQ	TPM1CH2	—
73	61	—	PTE6	KBI2P6	FB_D0	TXD3	—
74	62	—	PTE7	KBI2P7	$\overline{FB\_CS0}$	RXD3	—
75	63	—	PTJ0	FB_ALE	$\overline{FB\_CS1}$	—	—
76	64	—	PTJ1	—	FB_A4/FB_AD4	—	—
77	—	—	PTJ2	—	FB_A3/FB_AD3	—	—
78	—	—	PTJ3	—	FB_A2/FB_AD2	—	—
79	—	—	PTJ4	—	FB_A1/FB_AD1	—	—
80	—	—	PTJ5	—	FB_A0/FB_AD0	—	—

Keyboard interface functions associated with GPIO Ports E and G can be used whenever those pins are programmed for use as GPIO OR any other digital function.

GPIO input functions can be used when a pad is enabled for any digital function. This can be used to implement software-based debounce for KBI and IRQ functions.

Drive strength, pull-up enable<sup>1</sup>, slew rate and input filter settings in the GPIO apply to any digital use of the associated pin.

## 2.2.1 Peripheral Pinout Summary

The following tables break out pin options on a peripheral by peripheral basis.

**Table 2-2. Power/Ground Pinout Summary**

<b>80-Pin</b>	<b>64-Pin</b>	<b>48-Pin</b>	<b>Signals</b>
1	1	1	VDD1
21	17	13	VDD2
41	33	25	VDD3

1. There is one special case with regard to pull-up enable functions. PTC4 or PTE5 can be programmed to operate as IRQ. When in that mode, the pullup enable is controlled via IRQSC[IRQPD].

**Table 2-2. Power/Ground Pinout Summary (continued)**

61	49	37	VDD4
43	35	27	VDDA
2	2	2	VSS1
22	18	14	VSS2
42	34	26	VSS3
62	50	38	VSS4
44	36	28	VSSA

**Table 2-3. MII Pinout Summary**

80-Pin	64-Pin	48-Pin	Default Function	Signals	MII
30	26	22	PTC1	PTC1/MII_COL/SCL1	MII_COL
31	27	23	PTC2	PTC2/MII_CRS/SDA1	MII_CRS
5	5	5	PTA2	PTA2/MII_MDC/SCL2	MII_MDC
4	4	4	PTA1	PTA1/MII_MDIO/SDA2	MII_MDIO
11	11	11	PTB0	PTB0/MII_RX_CLK/SS2	MII_RX_CLK
10	10	10	PTA7	PTA7/MII_RX_DV/MOSI2	MII_RX_DV
12	12	12	PTB1	PTB1/MII_RX_ER/TMRCLK1	MII_RX_ER
9	9	9	PTA6	PTA6/MII_RXD0/MISO2	MII_RXD0
8	8	8	PTA5	PTA5/MII_RXD1/SPSCK2	MII_RXD1
7	7	7	PTA4	PTA4/MII_RXD2/RXD3	MII_RXD2
6	6	6	PTA3	PTA3/MII_RXD3/TXD3	MII_RXD3
24	20	16	PTB3	PTB3/MII_TX_CLK/MOSI1	MII_TX_CLK
25	21	17	PTB4	PTB4/MII_TX_EN/MISO1	MII_TX_EN
23	19	15	PTB2	PTB2/MII_TX_ER/SS1	MII_TX_ER
26	22	18	PTB5	PTB5/MII_TXD0/SPSCK1	MII_TXD0
27	23	19	PTB6	PTB6/MII_TXD1/TPM2CH0	MII_TXD1
28	24	20	PTB7	PTB7/MII_TXD2/TPM2CH1	MII_TXD2
29	25	21	PTC0	PTC0/MII_TXD3/TPM2CH2	MII_TXD3

**Table 2-4. Mini-FlexBus Pinout Summary**

80-Pin	64-Pin	48-Pin	Default Function	Signals	Mini-FlexBus
80	—	—	PTJ5	PTJ5/FB_A0/FB_AD0	FB_A0/FB_AD0
79	—	—	PTJ4	PTJ4/FB_A1/FB_AD1	FB_A1/FB_AD1
39	—	—	PTH6	PTH6/FB_A10/FB_AD10/TPM2CH1	FB_A10/FB_AD10

**Table 2-4. Mini-FlexBus Pinout Summary (continued)**

<b>80-Pin</b>	<b>64-Pin</b>	<b>48-Pin</b>	<b>Default Function</b>	<b>Signals</b>	<b>Mini-FlexBus</b>
38	—	—	PTH5	PTH5/FB_A11/FB_AD11	FB_A11/FB_AD11
37	—	—	PTH4	PTH4/FB_A12/FB_AD12	FB_A12/FB_AD12
36	32	—	PTF7/RGPIO15	PTF7/RGPIO15/FB_A13/FB_AD13/TPM2CH2	FB_A13/FB_AD13
35	31	—	PTF6/RGPIO14	PTF6/RGPIO14/FB_A14/FB_AD14/TPM2CH1	FB_A14/FB_AD14
17	—	—	PTH0	PTH0/FB_A15/FB_AD15	FB_A15/FB_AD15
16	16	—	PTF3/RGPIO11	PTF3/RGPIO11/FB_A16/FB_AD16	FB_A16/FB_AD16
15	15	—	PTF2/RGPIO10	PTF2/RGPIO10/FB_A17/FB_AD17	FB_A17/FB_AD17
14	14	—	PTF1/RGPIO9	PTF1/RGPIO9/FB_A18/FB_AD18	FB_A18/FB_AD18
13	13	—	PTF0/RGPIO8	PTF0/RGPIO8/FB_A19/FB_AD19	FB_A19/FB_AD19
78	—	—	PTJ3	PTJ3/FB_A2/FB_AD2	FB_A2/FB_AD2
77	—	—	PTJ2	PTJ2/FB_A3/FB_AD3	FB_A3/FB_AD3
76	64	—	PTJ1	PTJ1/FB_A4/FB_AD4	FB_A4/FB_AD4
56	48	—	PTG3/KBI1P3	PTG3/KBI1P3/FB_A5/FB_AD5/SDA1	FB_A5/FB_AD5
55	47	—	PTG2/KBI1P2	PTG2/KBI1P2/FB_A6/FB_AD6/SCL1	FB_A6/FB_AD6
54	46	—	PTG1/KBI1P1	PTG1/KBI1P1/FB_A7/FB_AD7/SDA2	FB_A7/FB_AD7
53	45	—	PTG0/KBI1P0	PTG0/KBI1P0/FB_A8/FB_AD8/SCL2	FB_A8/FB_AD8
40	—	—	PTH7	PTH7/FB_A9/FB_AD9/TPM2CH2	FB_A9/FB_AD9
71	59	47	PTE4/KBI2P4	PTE4/KBI2P4/CLKOUT/TPM1CH1	CLKOUT
73	61	—	PTE6/KBI2P6	PTE6/KBI2P6/FB_D0/TXD3	FB_D0
60	—	—	PTG7/KBI1P7	PTG7/KBI1P7/FB_D1	FB_D1
59	—	—	PTG6/KBI1P6	PTG6/KBI1P6/FB_D2	FB_D2
58	—	—	PTG5/KBI1P5	PTG5/KBI1P5/FB_D3	FB_D3
34	30	—	PTF5/RGPIO13	PTF5/RGPIO13/FB_D4/TPM2CH0	FB_D4
33	29	—	PTF4/RGPIO12	PTF4/RGPIO12/FB_D5/TMRCLK2	FB_D5
20	—	—	PTH3	PTH3/FB_D6/TPM2CH0	FB_D6
19	—	—	PTH2	PTH2/FB_D7/TMRCLK1	FB_D7
74	62	—	PTE7/KBI2P7	PTE7/KBI2P7/FB_CS0/RXD3	FB_CS0
75	63	—	PTJ0	PTJ0/FB_CS1/FB_ALE	FB_CS1/FB_ALE
18	—	—	PTH1	PTH1/FB_OE	FB_OE
57	—	—	PTG4/KBI1P4	PTG4/KBI1P4/FB_RW	FB_RW

**Table 2-5. IIC1 Pinout Summary**

<b>80-Pin</b>	<b>64-Pin</b>	<b>48-Pin</b>	<b>Default Function</b>	<b>Signals</b>	<b>IIC1</b>
55	47	—	PTG2/KBI1P2	PTG2/KBI1P2/FB_A6/FB_AD6/SCL1	SCL1
30	26	22	PTC1	PTC1/MII_COL/SCL1	SCL1
56	48	—	PTG3/KBI1P3	PTG3/KBI1P3/FB_A5/FB_AD5/SDA1	SDA1
31	27	23	PTC2	PTC2/MII_CRS/SDA1	SDA1

**Table 2-6. IIC2 Pinout Summary**

<b>80-Pin</b>	<b>64-Pin</b>	<b>48-Pin</b>	<b>Default Function</b>	<b>Signals4</b>	<b>IIC2</b>
53	45	—	PTG0/KBI1P0	PTG0/KBI1P0/FB_A8/FB_AD8/SC_L2	SCL2
4	4	4	PTA1	PTA1/MII_MDIO/SDA2	SCL2
47	39	31	PTC6	PTC6/SCL2/MISO1/ADP9	SCL2
54	46	—	PTG1/KBI1P1	PTG1/KBI1P1/FB_A7/FB_AD7/SDA2	SDA2
5	5	5	PTA2	PTA2/MII_MDC/SCL2	SDA2
48	40	32	PTC7	PTC7/SDA2/SPSCK1/ADP8	SDA2

**Table 2-7. SPI1 Pinout Summary**

<b>80-Pin</b>	<b>64-Pin</b>	<b>48-Pin</b>	<b>Default Function</b>	<b>Signals</b>	<b>SPI1</b>
47	39	31	PTC6	PTC6/SCL2/MISO1/ADP9	MISO1
25	21	17	PTB4	PTB4/MII_TX_EN/MISO1	MISO1
24	20	16	PTB3	PTB3/MII_TX_CLK/MOSI1	MOSI1
46	38	30	PTC5	PTC5/MOSI1/ADP10	MOSI1
48	40	32	PTC7	PTC7/SDA2/SPSCK1/ADP8	SPSCK1
26	22	18	PTB5	PTB5/MII_TXD0/SPSCK1	SPSCK1
23	19	15	PTB2	PTB2/MII_TX_ER/SS1	SS1
45	37	29	PTC4	PTC4/IRQ/SS1/ADP11	SS1

**Table 2-8. SPI2 Pinout Summary**

<b>80-Pin</b>	<b>64-Pin</b>	<b>48-Pin</b>	<b>Default Function</b>	<b>Signals</b>	<b>SPI12</b>
9	9	9	PTA6	PTA6/MII_RXD0/MISO2	MISO2
67	55	43	PTE0/KBI2P0	PTE0/KBI2P0/MISO2/ADP2	MISO2

**Table 2-8. SPI2 Pinout Summary (continued)**

10	10	10	PTA7	PTA7/MII_RX_DV/MOSI2	MOSI2
68	56	44	PTE1/KBI2P1	PTE1/KBI2P1/MOSI2/ADP1	MOSI2
8	8	8	PTA5	PTA5/MII_RXD1/SPSCK2	SPSCK2
66	54	42	PTD7GPIO7	PTD7GPIO7/SPSCK2/AD P3	SPSCK2
11	11	11	PTB0	PTB0/MII_RX_CLK/SS2	SS2
69	57	45	PTE2/KBI2P2	PTE2/KBI2P2/SS2/ADP0	SS2

**Table 2-9. SCI1 Pinout Summary**

80-Pin	64-Pin	48-Pin	Default Function	Signals	SCI1
50	42	34	PTD1/GPIO1	PTD1/GPIO1/RXD1/ADP6	RXD1
49	41	33	PTD0/GPIO0	PTD0/GPIO0/TXD1/ADP7	TXD1

**Table 2-10. SCI2 Pinout Summary**

80-Pin	64-Pin	48-Pin	Default Function	Signals	SCI12
52	44	36	PTD3/GPIO3	PTD3/GPIO3/RXD2/ADP4	RXD2
51	43	35	PTD2/GPIO2	PTD2/GPIO2/TXD2/ADP5	TXD2

**Table 2-11. SCI3 Pinout Summary**

80-Pin	64-Pin	48-Pin	Default Function	Signals	SCI3
74	62	—	PTE7/KBI2P7	PTE7/KBI2P7/FB_CS0/RXD3	RXD3
7	7	7	PTA4	PTA4/MII_RXD2/RXD3	RXD3
73	61	—	PTE6/KBI2P6	PTE6/KBI2P6/D0/TXD3	TXD3
6	6	6	PTA3	PTA3/MII_RXD3/TXD3	TXD3

**Table 2-12. TPM1 Pinout Summary**

80-Pin	64-Pin	48-Pin	Default Function	Signals4	TMR1
19	—	—	PTH2	PTH2/FB_D7/TMRCLK1	TMRCLK1
12	12	12	PTB1	PTB1/MII_RX_ER/TMRCLK1	TMRCLK1
33	29	—	PTF4/GPIO12	PTF4/GPIO12/FB_D5/TMRCLK2	TMRCLK2
70	58	46	PTE3/KBI2P3	PTE3/KBI2P3/TPM1CH0	TPM1CH0

**Table 2-12. TPM1 Pinout Summary (continued)**

71	59	47	PTE4/KBI2P4	PTE4/KBI2P4/CLKOUT/TPM1CH1	TPM1CH1
72	60	48	PTE5/KBI2P5	PTE5/KBI2P5/IRQ/TPM1CH2	TPM1CH2

**Table 2-13. TPM2 Pinout Summary**

80-Pin	64-Pin	48-Pin	Default Function	Signals	TMR2
19	—	—	PTH2	PTH2/FB_D7/TMRCLK1	TMRCLK1
12	12	12	PTB1	PTB1/MII_RX_ER/TMRCLK1	TMRCLK1
33	29	—	PTF4/RGPIO12	PTF4/RGPIO12/FB_D5/TMRCLK2	TMRCLK2
34	30	—	PTF5/RGPIO13	PTF5/RGPIO13/FB_D4/TPM2CH0	TPM2CH0
20	—	—	PTH3	PTH3/FB_D6/TPM2CH0	TPM2CH0
27	23	19	PTB6	PTB6/MII_TXD1/TPM2CH0	TPM2CH0
39	—	—	PTH6	PTH6/FB_A10/FB_AD10/TPM2CH1	TPM2CH1
35	31	—	PTF6/RGPIO14	PTF6/RGPIO14/FB_A14/FB_AD14/TPM2CH1	TPM2CH1
28	24	20	PTB7	PTB7/MII_TXD2/TPM2CH1	TPM2CH1
36	32	—	PTF7/RGPIO15	PTF7/RGPIO15/FB_A13/FB_AD13/TPM2CH2	TPM2CH2
40	—	—	PTH7	PTH7/FB_A9/FB_AD9/TPM2CH2	TPM2CH2
29	25	21	PTC0	PTC0/MII_TXD3/TPM2CH2	TPM2CH2

**Table 2-14. MTIM1 & MTIM2 Pinout Summary**

80-Pin	64-Pin	48-Pin	Default Function	Signals	MTIM1 & MTIM2
19	—	—	PTH2	PTH2/FB_D7/TMRCLK1	TMRCLK1
12	12	12	PTB1	PTB1/MII_RX_ER/TMRCLK1	TMRCLK1
33	29	—	PTF4/RGPIO12	PTF4/RGPIO12/FB_D5/TMRCLK2	TMRCLK2

**Table 2-15. ADC Pinout Summary**

80-Pin	64-Pin	48-Pin	Default Function	Signals	ADC
69	57	45	PTE2/KBI2P2	PTE2/KBI2P2/SS2/ADP0	ADP0
68	56	44	PTE1/KBI2P1	PTE1/KBI2P1/MOSI2/ADP1	ADP1
67	55	43	PTE0/KBI2P0	PTE0/KBI2P0/MISO2/ADP2	ADP2

**Table 2-15. ADC Pinout Summary (continued)**

66	54	42	PTD7/RGPIO7	PTD7/RGPIO7/SPSCK2/ADP3	ADP3
52	44	36	PTD3/RGPIO3	PTD3/RGPIO3/RXD2/ADP4	ADP4
51	43	35	PTD2/RGPIO2	PTD2/RGPIO2/TXD2/ADP5	ADP5
50	42	34	PTD1/RGPIO1	PTD1/RGPIO1/RXD1/ADP6	ADP6
49	41	33	PTD0/RGPIO0	PTD0/RGPIO0/TXD1/ADP7	ADP7
48	40	32	PTC7	PTC7/SDA2/SPSCK1/ADP8	ADP8
47	39	31	PTC6	PTC6/SCL2/MISO1/ADP9	ADP9
46	38	30	PTC5	PTC5/MOSI1/ADP10	ADP10
45	37	29	PTC4	PTC4/IRQ/SS1/ADP11	ADP11

**Table 2-16. GPIO Pinout Summary**

80-Pin	64-Pin	48-Pin	Default Function	Signals	GPIO
3	3	3	PTA0	PTA0/PHYCLK	PTA0
4	4	4	PTA1	PTA1/MII_MDIO/SDA2	PTA1
5	5	5	PTA2	PTA2/MII_MDC/SCL2	PTA2
6	6	6	PTA3	PTA3/MII_RXD3/TXD3	PTA3
7	7	7	PTA4	PTA4/MII_RXD2/RXD3	PTA4
8	8	8	PTA5	PTA5/MII_RXD1/SPSCK2	PTA5
9	9	9	PTA6	PTA6/MII_RXD0/MISO2	PTA6
10	10	10	PTA7	PTA7/MII_RX_DV/MOSI2	PTA7
11	11	11	PTB0	PTB0/MII_RX_CLK/SS2	PTB0
12	12	12	PTB1	PTB1/MII_RX_ER/TMRCLK1	PTB1
23	19	15	PTB2	PTB2/MII_TX_ER/SS1	PTB2
24	20	16	PTB3	PTB3/MII_TX_CLK/MOSI1	PTB3
25	21	17	PTB4	PTB4/MII_TX_EN/MISO1	PTB4
26	22	18	PTB5	PTB5/MII_RXD0/SPSCK1	PTB5
27	23	19	PTB6	PTB6/MII_RXD1/TPM2CH0	PTB6
28	24	20	PTB7	PTB7/MII_RXD2/TPM2CH1	PTB7
29	25	21	PTC0	PTC0/MII_RXD3/TPM2CH2	PTC0
30	26	22	PTC1	PTC1/MII_COL/SCL1	PTC1
31	27	23	PTC2	PTC2/MII_CRS/SDA1	PTC2
32	28	24	RESET	RESET/PTC3	PTC3
45	37	29	PTC4	PTC4/IRQ/SS1/ADP11	PTC4

**Table 2-16. GPIO Pinout Summary (continued)**

<b>80-Pin</b>	<b>64-Pin</b>	<b>48-Pin</b>	<b>Default Function</b>	<b>Signals</b>	<b>GPIO</b>
46	38	30	PTC5	PTC5/MOSI1/ADP10	PTC5
47	39	31	PTC6	PTC6/SCL2/MISO1/ADP9	PTC6
48	40	32	PTC7	PTC7/SDA2/SPSCK1/ADP8	PTC7
49	41	33	PTD0/RGPIO0	PTD0/RGPIO0/TXD1/ADP7	PTD0
50	42	34	PTD1/RGPIO1	PTD1/RGPIO1/RXD1/ADP6	PTD1
51	43	35	PTD2/RGPIO2	PTD2/RGPIO2/TXD2/ADP5	PTD2
52	44	36	PTD3/RGPIO3	PTD3/RGPIO3/RXD2/ADP4	PTD3
63	51	39	PTD4/RGPIO4	PTD4/RGPIO4/EXTAL	PTD4
64	52	40	PTD5/RGPIO5	PTD5/RGPIO5/XTAL	PTD5
65	53	41	BKGD/MS	BKGD/MS/PTD6/RGPIO6	PTD6
66	54	42	PTD7/RGPIO7	PTD7/RGPIO7/SPSCK2/ADP3	PTD7
67	55	43	PTE0/KBI2P0	PTE0/KBI2P0/MISO2/ADP2	PTE0
68	56	44	PTE1/KBI2P1	PTE1/KBI2P1/MOSI2/ADP1	PTE1
69	57	45	PTE2/KBI2P2	PTE2/KBI2P2/SS2/ADP0	PTE2
70	58	46	PTE3/KBI2P3	PTE3/KBI2P3/TPM1CH0	PTE3
71	59	47	PTE4/KBI2P4	PTE4/KBI2P4/CLKOUT/TPM1CH1	PTE4
72	60	48	PTE5/KBI2P5	PTE5/KBI2P5/IRQ/TPM1CH2	PTE5
73	61	—	PTE6/KBI2P6	PTE6/KBI2P6/FB_D0/TXD3	PTE6
74	62	—	PTE7/KBI2P7	PTE7/KBI2P7/FB_CS0/RXD3	PTE7
13	13	—	PTF0/RGPIO8	PTF0/RGPIO8/FB_A19/FB_AD19	PTF0
14	14	—	PTF1/RGPIO9	PTF1/RGPIO9/FB_A18/FB_AD18	PTF1
15	15	—	PTF2/RGPIO10 0	PTF2/RGPIO10/FB_A17/FB_AD17	PTF2
16	16	—	PTF3/RGPIO11 1	PTF3/RGPIO11/FB_A16/FB_AD16	PTF3
33	29	—	PTF4/RGPIO12 2	PTF4/RGPIO12/FB_D5/TMRCLK2	PTF4
34	30	—	PTF5/RGPIO13 3	PTF5/RGPIO13/FB_D4/TPM2CH0	PTF5
35	31	—	PTF6/RGPIO14 4	PTF6/RGPIO14/FB_A14/FB_AD14/TPM2CH1	PTF6
36	32	—	PTF7/RGPIO15 5	PTF7/RGPIO15/FB_A13/FB_AD13/TPM2CH2	PTF7
53	45	—	PTG0/KBI1P0	PTG0/KBI1P0/FB_A8/FB_AD8/SCL2	PTG0

**Table 2-16. GPIO Pinout Summary (continued)**

<b>80-Pin</b>	<b>64-Pin</b>	<b>48-Pin</b>	<b>Default Function</b>	<b>Signals</b>	<b>GPIO</b>
54	46	—	PTG1/KBI1P1	PTG1/KBI1P1/FB_A7/FB_AD7/SDA2	PTG1
55	47	—	PTG2/KBI1P2	PTG2/KBI1P2/FB_A6/FB_AD6/SCL1	PTG2
56	48	—	PTG3/KBI1P3	PTG3/KBI1P3/FB_A5/FB_AD5/SDA1	PTG3
57	—	—	PTG4/KBI1P4	PTG4/KBI1P4/FB_RW	PTG4
58	—	—	PTG5/KBI1P5	PTG5/KBI1P5/FB_D3	PTG5
59	—	—	PTG6/KBI1P6	PTG6/KBI1P6/FB_D2	PTG6
60	—	—	PTG7/KBI1P7	PTG7/KBI1P7/FB_D1	PTG7
17	—	—	PTH0	PTH0/FB_A15/FB_AD15	PTH0
18	—	—	PTH1	PTH1/FB_OE	PTH1
19	—	—	PTH2	PTH2/FB_D7/TMRCLK1	PTH2
20	—	—	PTH3	PTH3/FB_D6/TPM2CH0	PTH3
37	—	—	PTH4	PTH4/FB_A12/FB_AD12	PTH4
38	—	—	PTH5	PTH5/FB_A11/FB_AD11	PTH5
39	—	—	PTH6	PTH6/FB_A10/FB_AD10/TPM2CH1	PTH6
40	—	—	PTH7	PTH7/FB_A9/FB_AD9/TPM2CH2	PTH7
75	63	—	PTJ0	PTJ0/FB_CS1/FB_ALE	PTJ0
76	64	—	PTJ1	PTJ1/FB_A4/FB_AD4	PTJ1
77	—	—	PTJ2	PTJ2/FB_A3/FB_AD3	PTJ2
78	—	—	PTJ3	PTJ3/FB_A2/FB_AD2	PTJ3
79	—	—	PTJ4	PTJ4/FB_A1/FB_AD1	PTJ4
80	—	—	PTJ5	PTJ5/FB_A0/FB_AD0	PTJ5

**Table 2-17. KBI1 Pinout Summary**

<b>80-Pin</b>	<b>64-Pin</b>	<b>48-Pin</b>	<b>Default Function</b>	<b>Signals</b>	<b>KBI1</b>
53	45	—	PTG0/KBI1P0	PTG0/KBI1P0/FB_A8/FB_AD8/SC_L2	KBI1P0
54	46	—	PTG1/KBI1P1	PTG1/KBI1P1/FB_A7/FB_AD7/SD_A2	KBI1P1
55	47	—	PTG2/KBI1P2	PTG2/KBI1P2/FB_A6/FB_AD6/SC_L1	KBI1P2
56	48	—	PTG3/KBI1P3	PTG3/KBI1P3/FB_A5/FB_AD5/SD_A1	KBI1P3
57	—	—	PTG4/KBI1P4	PTG4/KBI1P4/FB_RW	KBI1P4

**Table 2-17. KBI1 Pinout Summary (continued)**

<b>80-Pin</b>	<b>64-Pin</b>	<b>48-Pin</b>	<b>Default Function</b>	<b>Signals</b>	<b>KBI1</b>
58	—	—	PTG5/KBI1P5	PTG5/KBI1P5/FB_D3	KBI1P5
59	—	—	PTG6/KBI1P6	PTG6/KBI1P6/FB_D2	KBI1P6
60	—	—	PTG7/KBI1P7	PTG7/KBI1P7/FB_D1	KBI1P7

**Table 2-18. KBI2 Pinout Summary**

<b>80-Pin</b>	<b>64-Pin</b>	<b>48-Pin</b>	<b>Default Function</b>	<b>Signals</b>	<b>KBI2</b>
67	55	43	PTE0/KBI2P0	PTE0/KBI2P0/MISO2/ADP2	KBI2P0
68	56	44	PTE1/KBI2P1	PTE1/KBI2P1/MOSI2/ADP1	KBI2P1
69	57	45	PTE2/KBI2P2	PTE2/KBI2P2/SS2/ADP0	KBI2P2
70	58	46	PTE3/KBI2P3	PTE3/KBI2P3/TPM1CH0	KBI2P3
71	59	47	PTE4/KBI2P4	PTE4/KBI2P4/CLKOUT/TPM1CH1	KBI2P4
72	60	48	PTE5/KBI2P5	PTE5/KBI2P5/IRQ/TPM1CH2	KBI2P5
73	61	—	PTE6/KBI2P6	PTE6/KBI2P6/FB_D0/TXD3	KBI2P6
74	62	—	PTE7/KBI2P7	PTE7/KBI2P7/FB_CS0/RXD3	KBI2P7

**Table 2-19. GPIO Pinout Summary**

<b>80-Pin</b>	<b>64-Pin</b>	<b>48-Pin</b>	<b>Default Function</b>	<b>Signals</b>	<b>GPIO</b>
49	41	33	PTD0/GPIO0	PTD0/GPIO0/TXD1/ADP7	GPIO0
50	42	34	PTD1/GPIO1	PTD1/GPIO1/RXD1/ADP6	GPIO1
15	15	—	PTF2/GPIO10	PTF2/GPIO10/FB_A17/FB_AD17	GPIO10
16	16	—	PTF3/GPIO11	PTF3/GPIO11/FB_A16/FB_AD16	GPIO11
33	29	—	PTF4/GPIO12	PTF4/GPIO12/FB_D5/TMRCLK2	GPIO12
34	30	—	PTF5/GPIO13	PTF5/GPIO13/FB_D4/TPM2CH0	GPIO13
35	31	—	PTF6/GPIO14	PTF6/GPIO14/FB_A14/FB_AD14/TPM2C_H1	GPIO14
36	32	—	PTF7/GPIO15	PTF7/GPIO15/FB_A13/FB_AD13/TPM2C_H2	GPIO15
51	43	35	PTD2/GPIO2	PTD2/GPIO2/TXD2/ADP5	GPIO2
52	44	36	PTD3/GPIO3	PTD3/GPIO3/RXD2/ADP4	GPIO3

**Table 2-19. GPIO Pinout Summary (continued)**

<b>80-Pin</b>	<b>64-Pin</b>	<b>48-Pin</b>	<b>Default Function</b>	<b>Signals</b>	<b>GPIO</b>
63	51	39	PTD4/RGPI04	PTD4/RGPI04/EXTAL	RGPI04
64	52	40	PTD5/RGPI05	PTD5/RGPI05/XTAL	RGPI05
65	53	41	BKGD/MS	BKGD/MS/PTD6/RGPI06	RGPI06
66	54	42	PTD7/RGPI07	PTD7/RGPI07/SPSCK2/ADP3	RGPI07
13	13	—	PTF0/RGPI08	PTF0/RGPI08/FB_A19/FB_AD19	RGPI08

## 2.3 Pin Mux Controls

Package pins on the MCF51CN128 can be programmed for up to four different functions using the Pin Mux Control Registers. Controls are organized by GPIO Port. Each GPIO port has two mux control registers, comprised of 2 bits per package pin. All pin mux registers reset to 0x00. Alternate values are defined in the remainder of this section. The encoding for each function matches the column number in which that function occurs in [Table 2-1](#). That is, default functions are assigned value 0x00, ALT1 functions 0x01, etc.

**Table 2-20. Pin Mux Control Registers**

<b>Address</b>	<b>Periphera l</b>	<b>Register</b>	<b>Bit 7</b>	<b>6</b>	<b>5</b>	<b>4</b>	<b>3</b>	<b>2</b>	<b>1</b>	<b>Bit 0</b>
0x(FF)FF_80C 0	MC	PTAPF1	A7		A6		A5		A4	
0x(FF)FF_80C 1	MC	PTAPF2	A3		A2		A1		A0	
0x(FF)FF_80C 2	MC	PTBPF1	B7		B6		B5		B4	
0x(FF)FF_80C 3	MC	PTBPF2	B3		B2		B1		B0	
0x(FF)FF_80C 4	MC	PTCPF1	C7		C6		C5		C4	
0x(FF)FF_80C 5	MC	PTCPF2	C3		C2		C1		C0	
0x(FF)FF_80C 6	MC	PTDPF1	D7		D6		D5		D4	
0x(FF)FF_80C 7	MC	PTDPF2	D3		D2		D1		D0	
0x(FF)FF_80C 8	MC	PTEPF1	E7		E6		E5		E4	
0x(FF)FF_80C 9	MC	PTEPF2	E3		E2		E1		E0	

**Table 2-20. Pin Mux Control Registers**

0x(FF)FF_80C A	MC	PTFPF1	F7		F6		F5	F4
0x(FF)FF_80C B	MC	PTFPF2	F8		F2		F1	F0
0x(FF)FF_80C C	MC	PTGPF1	G7		G6		G5	G4
0x(FF)FF_80C D	MC	PTGPF2	G3		G2		G1	G0
0x(FF)FF_80C E	MC	PTHPF1	H7		H6		H5	H4
0x(FF)FF_80C F	MC	PTHPF2	H3		H2		H1	H0
0x(FF)FF_80D 0	MC	PTJPF1	0	0	0	0	J5	J4
0x(FF)FF_80D 1	MC	PTJPF2	J3		J2		J1	J0

**Table 2-21. Pin Mux Control Register Bits**

Register Bit	Mux Control
Port PTA0 Pin Mux Controls	
00	PTA0
01	PHYCLK
10	Reserved
11	Reserved
Port PTA1 Pin Mux Controls	
00	PTA1
01	MII_MDIO
10	Reserved
11	SDA2
Port PTA2 Pin Mux Controls	
00	PTA2
01	MII_MDC
10	Reserved
11	SCL2
Port PTA3 Pin Mux Controls	
00	PTA3
01	MII_RXD3
10	TXD3
11	Reserved
Port PTA4 Pin Mux Controls	
00	PTA4
01	MII_RXD2
10	RXD3
11	Reserved
Port PTA5 Pin Mux Controls	
00	PTA5
01	MII_RXD1
10	SPSCK2
11	Reserved
Port PTA6 Pin Mux Controls	
00	PTA6
01	MII_RXD0

**Table 2-21. Pin Mux Control Register Bits (continued)**

<b>Register Bit</b>	<b>Mux Control</b>
10	MISO2
11	Reserved
Port PTA7 Pin Mux Controls	
00	PTA7
01	MII_RX_DV
10	MOSI2
11	Reserved
Port PTB0 Pin Mux Controls	
00	PTB0
01	MII_RX_CLK
10	SS2
11	Reserved
Port PTB1 Pin Mux Controls	
00	PTB1
01	MII_RX_ER
10	Reserved
11	TMRCLK1
Port PTB2 Pin Mux Controls	
00	PTB2
01	MII_TX_ER
10	SS1
11	Reserved
Port PTB3 Pin Mux Controls	
00	PTB3
01	MII_TX_CLK
10	MOSI1
11	Reserved
Port PTB4 Pin Mux Controls	
00	PTB4
01	MII_TX_EN
10	MISO1
11	Reserved
Port PTB5 Pin Mux Controls	

**Table 2-21. Pin Mux Control Register Bits (continued)**

<b>Register Bit</b>	<b>Mux Control</b>
00	PTB5
01	MII_TXD0
10	SPSCK1
11	Reserved
Port PTB6 Pin Mux Controls	
00	PTB6
01	MII_TXD1
10	Reserved
11	TPM2CH0
Port PTB7 Pin Mux Controls	
00	PTB7
01	MII_TXD2
10	Reserved
11	TPM2CH1
Port PTC0 Pin Mux Controls	
00	PTC0
01	MII_TXD3
10	Reserved
11	TPM2CH2
Port PTC1 Pin Mux Controls	
00	PTC1
01	MII_COL
10	Reserved
11	SCL1
Port PTC2 Pin Mux Controls	
00	PTC2
01	MII_CRS
10	Reserved
11	SDA1
Port PTC3 Pin Mux Controls	
00	RESET
01	PTC3
10	Reserved

**Table 2-21. Pin Mux Control Register Bits (continued)**

<b>Register Bit</b>	<b>Mux Control</b>
11	Reserved
Port PTC4 Pin Mux Controls	
00	PTC4
01	IRQ
10	SS1
11	ADP11
Port PTC5 Pin Mux Controls	
00	PTC5
01	Reserved
10	MOSI1
11	ADP10
Port PTC6 Pin Mux Controls	
00	PTC6
01	SCL2
10	MISO1
11	ADP9
Port PTC7 Pin Mux Controls	
00	PTC7
01	SDA2
10	SPSCK1
11	ADP8
Port PTD0 Pin Mux Controls	
00	PTD0/RGPIO0
01	Reserved
10	TXD1
11	ADP7
Port PTD1 Pin Mux Controls	
00	PTD1/RGPIO1
01	Reserved
10	RXD1
11	ADP6
Port PTD2 Pin Mux Controls	
00	PTD2/RGPIO2

**Table 2-21. Pin Mux Control Register Bits (continued)**

<b>Register Bit</b>	<b>Mux Control</b>
01	Reserved
10	TXD2
11	ADP5
Port PTD3 Pin Mux Controls	
00	PTD3/RGPIO3
01	Reserved
10	RXD2
11	ADP4
Port PTD4 Pin Mux Controls	
00	PTD4/RGPIO4
01	Reserved
10	Reserved
11	EXTAL
Port PTD5 Pin Mux Controls	
00	PTD5/RGPIO5
01	Reserved
10	Reserved
11	XTAL
Port PTD6 Pin Mux Controls	
00	BKGD/MS
01	PTD6/RGPIO6
10	Reserved
11	Reserved
Port PTD7 Pin Mux Controls	
00	PTD7RGPIO7
01	Reserved
10	SPSCK2
11	ADP3
Port PTE0 Pin Mux Controls	
00	PTE0
01	KBI2P0
10	MISO2
11	ADP2

**Table 2-21. Pin Mux Control Register Bits (continued)**

<b>Register Bit</b>	<b>Mux Control</b>
Port PTE1 Pin Mux Controls	
00	PTE1
01	KBI2P1
10	MOSI2
11	ADP1
Port PTE2 Pin Mux Controls	
00	PTE2
01	KBI2P2
10	$\overline{SS2}$
11	ADP0
Port PTE3 Pin Mux Controls	
00	PTE3
01	KBI2P3
10	Reserved
11	TPM1CH0
Port PTE4 Pin Mux Controls	
00	PTE4
01	KBI2P4
10	CLKOUT
11	TPM1CH1
Port PTE5 Pin Mux Controls	
00	PTE5
01	KBI2P5
10	IRQ
11	TPM1CH2
Port PTE6 Pin Mux Controls	
00	PTE6
01	KBI2P6
10	FB_D0
11	TXD3
Port PTE7 Pin Mux Controls	
00	PTE7
01	KBI2P7

**Table 2-21. Pin Mux Control Register Bits (continued)**

<b>Register Bit</b>	<b>Mux Control</b>
10	FB_CS0
11	RXD3
Port PTF0 Pin Mux Controls	
00	PTF0/RGPIO8
01	Reserved
10	FB_A19/FB_AD19
11	Reserved
Port PTF1 Pin Mux Controls	
00	PTF1/RGPIO9
01	Reserved
10	FB_A18/FB_AD18
11	Reserved
Port PTF2 Pin Mux Controls	
00	PTF2/RGPIO10
01	Reserved
10	FB_A17/FB_AD17
11	Reserved
Port PTF3 Pin Mux Controls	
00	PTF3/RGPIO11
01	Reserved
10	FB_A16/FB_AD16
11	Reserved
Port PTF4 Pin Mux Controls	
00	PTF4/RGPIO12
01	Reserved
10	FB_D5
11	TMRCLK2
Port PTF5 Pin Mux Controls	
00	PTF5/RGPIO13
01	Reserved
10	FB_D4
11	TPM2CH0
Port PTF6 Pin Mux Controls	

**Table 2-21. Pin Mux Control Register Bits (continued)**

<b>Register Bit</b>	<b>Mux Control</b>
00	PTF6/RGPIO14
01	Reserved
10	FB_A14/FB_AD14
11	TPM2CH1
Port PTF7 Pin Mux Controls	
00	PTF7/RGPIO15
01	Reserved
10	FB_A13/FB_AD13
11	TPM2CH2
Port PTG0 Pin Mux Controls	
00	PTG0
01	KBI1P0
10	FB_A8/FB_AD8
11	SCL2
Port PTG1 Pin Mux Controls	
00	PTG1
01	KBI1P1
10	FB_A7/FB_AD7
11	SDA2
Port PTG2 Pin Mux Controls	
00	PTG2
01	KBI1P2
10	FB_A6/FB_AD6
11	SCL1
Port PTG3 Pin Mux Controls	
00	PTG3
01	KBI1P3
10	FB_A5/FB_AD5
11	SDA1
Port PTG4 Pin Mux Controls	
00	PTG4
01	KBI1P4
10	FB_R <sup>W</sup>

**Table 2-21. Pin Mux Control Register Bits (continued)**

<b>Register Bit</b>	<b>Mux Control</b>
11	Reserved
Port PTG5 Pin Mux Controls	
00	PTG5
01	KBI1P5
10	FB_D3
11	Reserved
Port PTG6 Pin Mux Controls	
00	PTG6
01	KBI1P6
10	FB_D2
11	Reserved
Port PTG7 Pin Mux Controls	
00	PTG7
01	KBI1P7
10	FB_D1
11	Reserved
Port PTH0 Pin Mux Controls	
00	PTH0
01	Reserved
10	FB_A15/FB_AD15
11	Reserved
Port PTH1 Pin Mux Controls	
00	PTH1
01	Reserved
10	FB_OE
11	Reserved
Port PTH2 Pin Mux Controls	
00	PTH2
01	Reserved
10	FB_D7
11	TMRCLK1
Port PTH3 Pin Mux Controls	
00	PTH3

**Table 2-21. Pin Mux Control Register Bits (continued)**

<b>Register Bit</b>	<b>Mux Control</b>
01	Reserved
10	FB_D6
11	TPM2CH0
Port PTH4 Pin Mux Controls	
00	PTH4
01	Reserved
10	FB_A12/FB_AD12
11	Reserved
Port PTH5 Pin Mux Controls	
00	PTH5
01	Reserved
10	FB_A11/FB_AD11
11	Reserved
Port PTH6 Pin Mux Controls	
00	PTH6
01	Reserved
10	FB_A10/FB_AD10
11	TPM2CH1
Port PTH7 Pin Mux Controls	
00	PTH7
01	Reserved
10	FB_A9/FB_AD9
11	TPM2CH2
Port PTJ0 Pin Mux Controls	
00	PTJ0
01	ALE
10	FB_CS1
11	Reserved
Port PTJ1 Pin Mux Controls	
00	PTJ1
01	Reserved
10	FB_A4/FB_AD4
11	Reserved

**Table 2-21. Pin Mux Control Register Bits (continued)**

Register Bit	Mux Control
Port PTJ2 Pin Mux Controls	
00	PTJ2
01	Reserved
10	FB_A3/FB_AD3
11	Reserved
Port PTJ3 Pin Mux Controls	
00	PTJ3
01	Reserved
10	FB_A2/FB_AD2
11	Reserved
Port PTJ4 Pin Mux Controls	
00	PTJ4
01	Reserved
10	FB_A1/FB_AD1
11	Reserved
Port PTJ5 Pin Mux Controls	
00	PTJ5
01	Reserved
10	FB_A0/FB_AD0
11	Reserved

## 2.4 Basic System Connections

Figure 2-4 shows pin connections that are common to MCF51CN128 series application systems.

### Figure NOTES:

1. The RESET pin can only reset the device into user mode. You can not enter BDM using the RESET pin. BDM can be entered by holding MS low during POR or writing a 1 to BDM\_RESET in the ColdFire XCSR register with MS low after issuing BDM command.
2. RESET/PTC3 features an optional internal pullup device.
3. RC filter on RESET/PTC3 pin recommended for noisy environments.

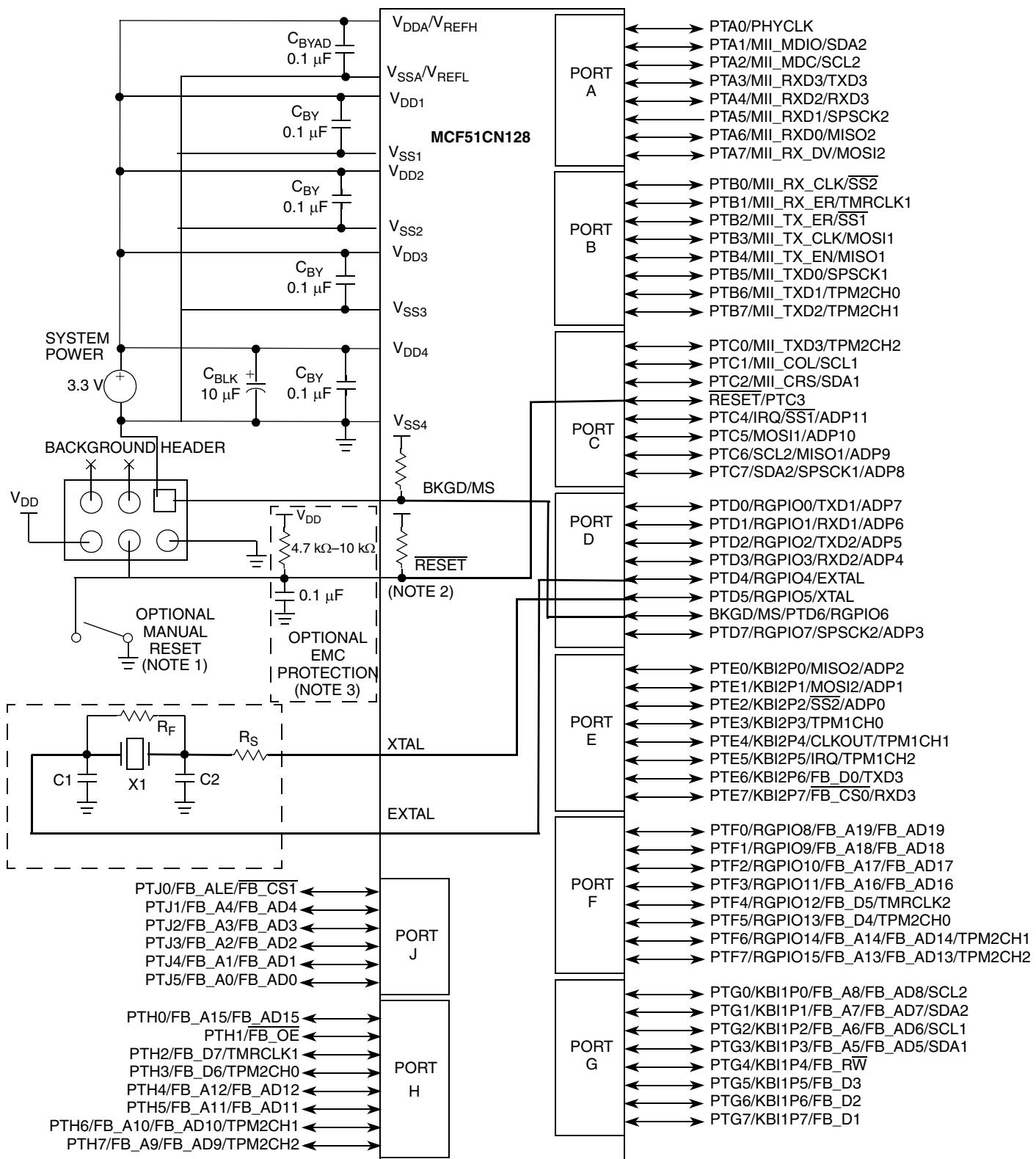


Figure 2-4. Basic System Connections

## 2.4.1 Power

$V_{DD1,2,3,4}$  and  $V_{SS1,2,3,4}$  are the primary power supply pins for the microcontroller. This voltage source supplies power to all I/O buffer circuitry and to an internal voltage regulator. The internal voltage regulator provides a regulated lower-voltage source to the CPU and other internal circuitry of the microcontroller.

Typically, application systems have two separate capacitor values across the power pins. In this case, there should be:

- A bulk electrolytic capacitor, such as a  $10\mu F$  tantalum capacitor, to provide bulk charge storage for the overall system.
- A  $0.1 \mu F$  ceramic bypass capacitor located as close to the microcontroller power pins as practical to suppress high-frequency noise. The MCF51CN128 has four  $V_{DD}$  pins. Each pin must have a bypass capacitor for best noise suppression.

$V_{DDA}$  and  $V_{SSA}$  are the analog power supply pins for the microcontroller. This voltage source supplies power to the ADC module. A  $0.1 \mu F$  ceramic bypass capacitor should be located as close to the microcontroller power pins as practical to suppress high-frequency noise.

## 2.4.2 Oscillator

Immediately after reset, the microcontroller uses an internally generated clock provided by the multipurpose clock generation (MCG) module.

The oscillator (XOSC) in this microcontroller is a Pierce oscillator that can accommodate a crystal or ceramic resonator. Optionally, an external clock source can be connected to the EXTAL input pin.

Refer to [Figure 2-4](#) for the following discussion.  $R_S$  (when used) and  $R_F$  should be low-inductance resistors such as carbon composition resistors. Wire-wound resistors, and some metal film resistors, have too much inductance.  $C_1$  and  $C_2$  normally should be high-quality ceramic capacitors that are specifically designed for high-frequency applications.

$R_F$  is used to provide a bias path to keep the EXTAL input in its linear range during crystal startup; its value is not generally critical. Typical systems use  $1 M\Omega$  to  $10 M\Omega$ . Higher values are sensitive to humidity and lower values reduce gain and (in extreme cases) could prevent startup.

$C_1$  and  $C_2$  are typically in the  $5 \text{ pF}$  to  $25 \text{ pF}$  range and are chosen to match the requirements of a specific crystal or resonator. Take into account printed circuit board (PCB) capacitance and microcontroller pin capacitance when selecting  $C_1$  and  $C_2$ . The crystal manufacturer typically specifies a load capacitance which is the series combination of  $C_1$  and  $C_2$  (which are usually the same size). As a first-order approximation, use  $10 \text{ pF}$  as an estimate of combined pin and PCB capacitance for each oscillator pin (EXTAL and XTAL).

### 2.4.3 RESET/PTC3

The RESET/PTC3 pin defaults to hardware reset upon a power-on-reset event. Unless otherwise programmed, it continues in that role indefinitely. It can also be programmed as an open drain GPIO output. It should not be programmed as a GPIO input, as an external driver on that pin could drive a zero on that pin during power up, which would prevent the part from exiting the power-on-reset sequence. During STOP2, the RESET/PTC3 pin can be used to wake the device from that state. When an application uses the STOP2 state, RESET/PTC3 must be pre-configured as RESET prior to entering STOP2. There is a direct analog connection from this pad to the power management controller wakeup pin. PTC3 configured as a GPIO output could prevent proper operation of STOP2.

Using RESET/PTC3 as RESET is optional, since internal power-on reset and low-voltage reset circuitry typically make external reset circuitry unnecessary. The internal pull-up on this pin is enabled upon any device reset.

RESET/PTC3 is normally connected to the standard 6-pin background debug connector so a development system can directly reset the MCU system. If desired, a manual external reset can be added by supplying a simple switch to ground (pull reset pin low to force a reset).

In EMC-sensitive applications, an external RC filter is recommended on this pin. See [Figure 2-4](#) for an example.

When any reset is initiated (whether from an external source or from an internal source, the RESET pin is driven low for approximately 66 bus cycles and released. The reset circuitry decodes the cause of reset and records it by setting a corresponding bit in the system control reset status register (SRS).

### 2.4.4 IRQ

The IRQ pin function acts as a non-maskable interrupt to the V1 ColdFire core. This function can be programmed to occur on either PTC4 or PTE5.

### 2.4.5 Background / Mode Select (BKGD/MS)

During a power-on-reset (POR) or background debug force reset (see bit ENBDM in [Section 20.3.2, “Extended Configuration/Status Register \(XCSR\),”](#) for more information), the BKGD/MS pin functions as a mode select pin. Immediately after any reset, the pin functions as the background pin and can be used for background debug communication. The internal pull-up on this pin is enabled upon any device reset.

If the BKGD/MS pin is unconnected, the microcontroller enters normal operating mode at the rising edge of the internal reset after a POR or forced BDC reset. If a debug system is connected to the 6-pin standard background debug header, it can hold BKGD/MS low during a POR or immediately after issuing a background debug force reset<sup>1</sup>, which forces the microcontroller into halt mode.

The BKGD/MS pin is used primarily for background debug controller (BDC) communications using a custom protocol that uses 16 clock cycles of the target microcontroller’s BDC clock per bit time. The target

1. Specifically, BKGD must be held low through the first 16 cycles after deassertion of the internal reset.

## Pins and Connections

microcontroller's BDC clock could be as fast as the bus clock rate, so there should never be any significant capacitance connected to the BKGD/MS pin that could interfere with background serial communications.

Although the BKGD/MS pin is a pseudo open-drain pin, the background debug communication protocol provides brief, actively driven, high speed-up pulses to ensure fast rise times. Small capacitances from cables and the absolute value of the internal pullup device play almost no role in determining rise and fall times on the BKGD/MS pin.

The BKGD/MS select pin can be reprogrammed to operate as one of the Rapid GPIO pins on this device, or as a standard GPIO. It should only be programmed for use as an output, as an external signal driving this pin during startup may cause the device to boot into debug mode.

### 2.4.6 ADC Reference Pins ( $V_{REFH}$ , $V_{REFL}$ )

$V_{REFH}$  and  $V_{REFL}$  are the voltage reference high and low inputs, respectively, for the ADC module. These are internally connected to  $V_{DDA}$  and  $V_{SSA}$ , and are not present as explicit pins on the device.

### 2.4.7 General-Purpose I/O and Peripheral Ports

The MCF51CN128 series microcontrollers support up to 70 general-purpose I/O pins<sup>1</sup>, which are shared with on-chip peripheral functions (timers, serial I/O, ADC, etc.).

When a port pin is configured as a general-purpose output or a peripheral uses the port pin as an output, software can select one of two drive strengths and enable or disable slew rate control.

When a port pin is configured as a general-purpose input or a peripheral uses the port pin as an input, software can enable a pull-up device. Pad cells on these devices also include an optional low pass filter in the inputs. Again, this can be enabled via software control.

Immediately after reset, these pins (excluding the RESET and BKGD/MS pins) are configured as high-impedance general-purpose inputs with internal pull-up devices disabled.

When an on-chip peripheral system is controlling a pin, data direction control bits still determine what is read from the port data registers, even though the peripheral controls the pin direction via the pin's output buffer enable. For information about controlling these pins as general-purpose I/O pins, see “[Chapter 9, “Parallel Input/Output Control”](#)”.

#### NOTE

To avoid extra current drain from floating input pins, the reset initialization routine in the application program should enable on-chip pullup devices or change the direction of unused pins to outputs so they do not float.

---

1. There are restrictions on the use of RESET and BKGD/MS as GPIO. See those sections for details.

# Chapter 3

## Modes of Operation

### 3.1 Introduction

The operating modes of the MCF51CN128 are described in this chapter. Entry into each mode, exit from each mode, and functionality while in each of the modes are described.

The overall system mode is generally a function of a number of separate, but inter-related variables: debug mode, security mode, power mode, and clock mode. Clock modes are discussed in [Section 1.4.4, “MCG Modes of Operation.”](#) This chapter explores the other dimensions of the system operating mode.

### 3.2 Features

**Table 3-1. MCF51CN128 Mode Features**

Mode	Description
Debug	Useful for code development. This device, like all V1 ColdFire devices, debug mode is mutually exclusive with use of secure mode.
Secure mode	BDC access to CPU resources is extremely restricted. It is possible to tell that the device has been secured, and to clear security, which involves mass erasing the on-chip flash memory. No other CPU access is allowed. Secure mode can be used in conjunction with each of the power modes below.
Run mode	CPU clocks can be run at full speed and the internal supply is fully regulated.
LPrun mode	CPU and peripheral clocks are restricted to 250 kHz CPU clock and 125 kHz bus clock maximum and the internal supply is in soft regulation.
Wait mode	CPU shuts down to conserve power; peripheral clocks are running and full regulation is maintained.
LPwait mode	CPU shuts down to conserve power; peripheral clocks are running at reduced speed (125 kHz maximum bus clock) and the internal voltage regulator is running in loose regulation mode.
Stop modes	System (CPU and peripheral) clocks are stopped. <ul style="list-style-type: none"><li>• Stop4 — All internal circuits are powered (full regulation mode) and internal clock sources still at max frequency for fastest recovery.</li><li>• Stop3 — All internal circuits are loosely regulated and clocks sources are at minimal values (125 kHz maximum bus clock), providing a good compromise between power utilization and speed of recovery.</li><li>• Stop2 — Partial power-down of internal circuits; RAM content is retained. The lowest power mode for this device. A reset is required to return from stop2 mode.</li></ul>

On the MCF51CN128, wait, stop2, stop3, and stop4 are entered with the CPU STOP instruction. See [Table 3-2, Figure 3-2](#), and subsequent sections of this chapter for details.

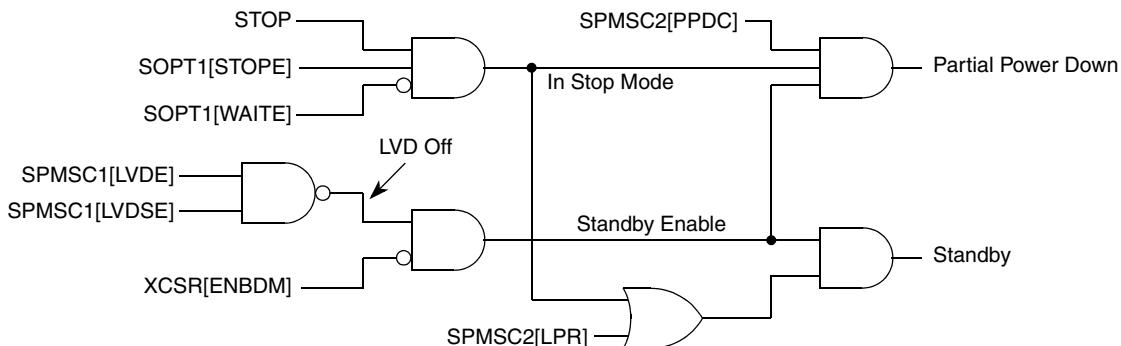
### 3.3 Overview

The ColdFire CPU has two primary user modes of operation: run and stop. (The CPU also supports a halt mode that is used strictly for debug operations.) The STOP instruction is used to invoke stop and wait modes for this family of devices.

The systems option register 1 (SOPT1) contains two bits which control operation of the STOP instruction. If SOPT1[WAITE] is set when STOP is executed, then wait mode is entered. Otherwise, if SOPT1[STOPE] is set, the CPU enters one of the stop modes. It is illegal to execute a STOP instruction if neither STOPE or WAITE are set. This results in reset assertion if the (CPUCR[IRD]) bit is cleared or an illegal instruction exception if CPUCR[IRD] is set.

The MCF51CN128 devices augment stop, wait, and run in a number of ways. The power management controller (PMC) can run the device in fully-regulated mode, standby mode, and partial power-down mode. Standby (loose regulation) or partial power-down can be programmed to occur naturally as a result of a STOP instruction. Additionally, standby mode can be explicitly invoked by the (SPMSC2[LPR]) bit. Use of standby is limited to bus frequencies less than 125 kHz; and neither standby nor partial power-down are allowed when XCSR[ENBDM] bit is set that enables debugging in stop and wait modes.

During partial power-down mode, the regulator is in standby mode and much of the digital logic on the device is switched off. These interactions can be seen schematically in [Figure 3-1](#). This figure is for conceptual purposes only. It does not reflect any sequence or time dependencies between the PMC and other parts of the device, nor does it represent any actual design partitioning.



**Figure 3-1. MCF51CN128 Power Modes - Conceptual Drawing**

It is illegal for the software to have SPMSC2[PPDC] and SPMSC2[LPR] asserted concurrently. This restriction arises because the sequence of events from normal to low-power modes involves using both bits. After entering a low-power mode, it is not possible to switch to another low-power mode.

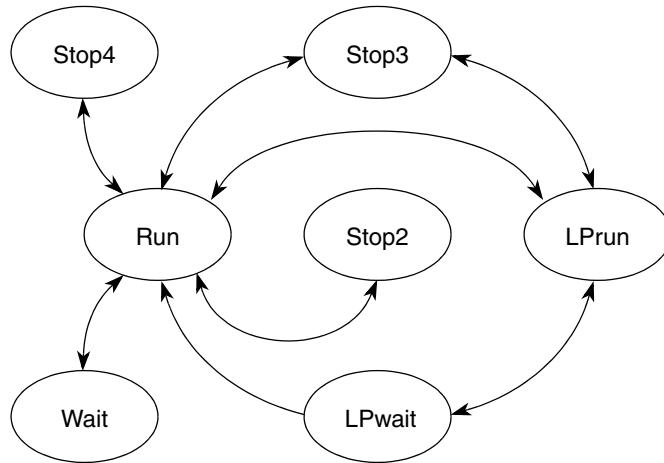
Table 3-2. CPU / Power Mode Selections

Mode of Operation	SOPT1 SIM		XCSR BDC	SPMSC1 PMC		SPMSC2 PMC		CPU and Peripheral Clocks	Effects on Sub-System	
	STOPE	WAITE	ENBDM <sup>1</sup>	LVDE	LVDSE	LPR	PPDC		BDC Clock	Switched Power
<b>Run mode</b> - processor and peripherals clocked normally.	x	x	x	x	x	0	x	On. MCG in any mode	On	On
			x	1	1	x	x			
			1	x	x	x	x			
<b>LPrun mode with low voltage detect disabled<sup>2</sup></b> - processor and peripherals clocked at low frequency <sup>3</sup> . Low voltage detects are not active.	x	x	0	0	x	1	0	Low freq required. MCG in BLPE mode.	Note: When not needed, the BDC clock can be gated off at the discretion of the processor. The clock is available within a few cycles of demand by the processor, normally when a negative edge is detected on BKGD. The BDM command associated with that negative edge may not take affect.	Loose Reg
			0	1	0					
<b>Wait mode</b> - processor clock nominally inactive, but peripherals are clocked.	x	1	x	x	x	0	x	Periph clocks on. CPU clock on if XCSR[ENBDM]=1 .	On	On
			x	1	1	x	x			
			1	x	x	x	x			
<b>LPwait mode</b> - processor clock is inactive, peripherals are clocked at low frequency and the PMC is loosely regulating. Low voltage detects are not active.	x	1	0	0	x	1	0	CPU clock is off. Periph clocks at low speed. MCG in BLPE.	CPU clock is off. Periph clocks at low speed. MCG in BLPE.	Loose Reg
				1	0					
<b>Stop modes disabled</b> ; Illegal opcode reset if STOP instruction executed and CPUCR[IRD] is cleared, else illegal instruction exception is generated.	0	0	Function of BKGD/MS at reset	1	1	0	0	On	Function of BKGD/MS at reset	On
<b>Stop4</b> - Either low-power modes have not been requested, or low voltage detects are enabled or XCSR[ENBDM] = 1.	1	0	x	x	x	0	0	Peripheral clocks off. CPU clock on if XCSR[ENBDM]=1 .	BDC clock enabled only if XCSR[ENBDM]=1 prior to entering stop.	On
			x	1	1	1	0			
			x	1	1	0	1			
			1	x	x	x	x	CPU clock on. Periph clocks off.		
<b>Stop3</b> - Low voltage detect in stop is not enabled. Clocks must be at low frequency and are gated. The regulator is in loose regulation.	1	0	0	1	0	X	0	Low freq required. MCG in BLPE mode. CPU clock off. Some peripheral clocks are optionally on. See Table 3-5 for details.	Off	Loose Reg
<b>Stop2</b> - Low voltage detects are not active. If BDC is enabled, stop4 is invoked rather than stop2.	1	0		0	x					
				1	0	0	1	N/A	N/A	Off
				0	x					

<sup>1</sup> ENBDM is located in the upper byte of the XCSR register which is write-accessible only through BDC commands. See [Section 20.3.2, "Extended Configuration/Status Register \(XCSR\)".](#)

<sup>2</sup> If a device is in low power run mode, a falling edge on an active BKGD/MS pin exits low power run mode, clears the LPRS bits and returns the device to normal run mode.

<sup>3</sup> In LPrun maximum 250 kHz CPU frequency and 125 kHz peripheral clock frequency.



Mode	Regulator State
Run	Full On
Wait	Full On
Stop4	Full On
LPrun	Standby
LPwait	Standby
Stop3	Standby
Stop2	Partial Power Off

**Figure 3-2. Allowable Power Mode Transitions for the MCF51CN128 Series**

Figure 3-2 illustrates mission mode state transitions allowed between the legal states shown in Table 3-2.  $\overline{\text{RESET}}$  must be asserted low or the RTC must issue a wakeup signal to exit stop2. Only an interrupt assertion is necessary to exit the other stop and wait modes.

Figure 3-3 takes the same set of states and transitions shown in Figure 3-2 and adds the BDM halt mode for development purposes. If BDM is enabled, the chip automatically shifts LP modes into their fully regulated equivalents. If software or debugger sets SPMSC2[LPR] while BDM is enabled, SPMSC2[LPRS] reflects that the regulator is not in standby. Similarly, SPMSC2[PPDF] does not indicate a recovery from stop2 if XCSR[ENBDM] forced stop4 to occur in its place.<sup>1</sup>

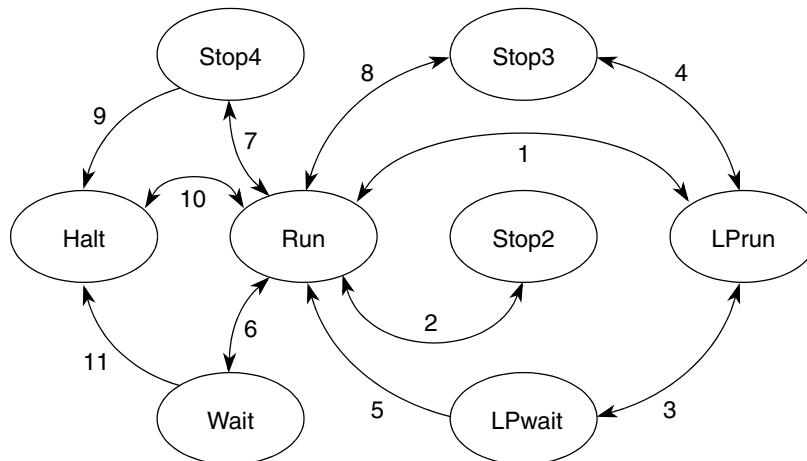
Stated another way, if XCSR[ENBDM] has been set via the BDM interface, then the power management controller keeps (or puts) the regulator in full regulation despite other settings in the contrary. The states shown in Figure 3-3 then map as follows:

- $\text{LPrun} \Rightarrow \text{Run}$
- $\text{LPwait} \Rightarrow \text{Wait}$
- $\text{Stop3} \Rightarrow \text{Stop4}$
- $\text{Stop2} \Rightarrow \text{Stop4}$

From a software perspective (and disregarding PMC status bits), the system remains in the appropriate low-power state, and can be debugged as such.

See Section 3.7, “Wait Modes,” for a description of the various methods to enter halt mode.

1. This can have subtle impacts on recovery from stop. The IRQ input can wake the device from stop4 if it has been enabled for that purpose. That same pin wakes the device from stop2 even when the IRQ is not enabled (there is an asynchronous path to the power management controller in that state).

**Figure 3-3. All Allowable Power Mode Transitions for MCF51CN128**

[Table 3-3](#) defines triggers for the various state transitions shown in [Figure 3-2](#).

**Table 3-3. Triggers for Transitions Shown in [Figure 3-2](#)**

Transition #	From	To	Trigger
1	Run	LPrun	Configure settings shown in <a href="#">Table 3-2</a> , switch LPR=1 last
	LPrun	Run	Clear SPMSC2[LPR]
			Interrupt when SPMSC2[LPWUI]=1
		Stop2	Negative transition on enabled BKGD/MS pin.
2	Run	Stop2	Pre-configure settings shown in <a href="#">Table 3-2</a> , execute STOP instruction
	Stop2	Run	Assert <u>RESET</u> /PTC3 <sup>1</sup> low or RTC timeout. Reload environment from RAM
3	LPrun	LPwait	Pre-configure settings shown in <a href="#">Table 3-2</a> , execute STOP instruction
	LPwait	LPrun	Interrupt when SPMSC2[LPWUI]=0
4	LPrun	Stop3	Execute STOP instruction
	Stop3	LPrun	Interrupt when SPMSC2[LPWUI]=0
5	LPwait	Run	Interrupt when SPMSC2[LPWUI]=1
	Run	LPwait	Not supported.
6	Run	Wait	Pre-configure settings shown in <a href="#">Table 3-2</a> , execute STOP instruction
	Wait	Run	Interrupt
7	Run	Stop4	Pre-configure settings shown in <a href="#">Table 3-2</a> , execute STOP instruction
	Stop4	Run	Interrupt

**Table 3-3. Triggers for Transitions Shown in Figure 3-2 (continued)**

Transition #	From	To	Trigger
8	Stop3	Run	Interrupt when SPMSC2[LPWUI] is set
	Run	Stop3	Pre-configure settings shown in <a href="#">Table 3-2</a> , execute STOP instruction
9	Stop4	Halt	When a BACKGROUND command is received through the BKGD/MS pin (XCSR[ENBDM] must be set).
	Halt	Stop4	Not supported.
10	Halt	Run	GO instruction issued via BDM
	Run	Halt	When a BACKGROUND command is received through the BKGD/MS pin OR When a HALT instruction is executed OR When encountering a BDM breakpoint
11	Wait	Halt	When a BACKGROUND command is received through the BKGD/MS pin (XCSR[ENBDM] must be set).
	Halt	Wait	Not supported.

<sup>1</sup> An analog connection from this pin to the on-chip regulator wakes the regulator, which then initiates a power-on-reset sequence.

Individual power states are discussed in more detail in the following sections.

## 3.4 Debug Mode

Debug mode functions are managed through the background debug controller (BDC) in the Version 1 ColdFire core. The BDC allows you to analyze MCU operation during software development.

The debug interface is used to write a bootloader or user application program into the flash program memory before the MCU operates in run mode for the first time. When the MCF51CN128 is shipped from the Freescale Semiconductor factory, the flash program memory is erased by default unless specifically noted. So, there is no program that can be executed in run mode until the flash memory is initially programmed. The debug interface can also be used to erase and reprogram the flash memory after it has been previously programmed.

See [Chapter 20, “Version 1 ColdFire Debug \(CF1\\_DEBUG\),”](#) for more details regarding the debug interface.

## 3.5 Secure Mode

While the MCU is in secure mode, there are severe restrictions on which debug commands can be used. In this mode, only the upper byte of the core’s XCSR, CSR2, and CSR3 registers can be accessed. See [Chapter 20, “Version 1 ColdFire Debug \(CF1\\_DEBUG\),”](#) for details.

## 3.6 Run Modes

### 3.6.1 Run Mode

Run mode is the normal operating mode for this device. This mode is selected when the BKGD/MS pin is high at the rising edge of the internal reset signal. Upon exiting reset, the CPU fetches the supervisor SR and initial PC from locations 0x(00)00\_0000 and 0x(00)00\_0004 and executes code starting at the newly set value of the PC.

### 3.6.2 Low-Power Run Mode (LPrun<sup>1</sup>)

In the low-power run mode, the on-chip voltage regulator is put into its standby (or loose regulation) state. In this state, the power consumption is reduced to a minimum that allows CPU functionality. To further reduce power consumption, disable the clocks to all unused peripherals by clearing the corresponding bits in SCGC1 - 4 registers<sup>2</sup>.

Before entering this mode, the following conditions must be met:

- BLPE<sup>3</sup> is the selected clock mode for the MCG. See [Section 6.4.1, “MCG Modes of Operation,”](#) for more details.
- The bus frequency is less than 125 kHz.
- The ADC is in low-power mode (ADCCFG[ADLPC]=1) or disabled.
- Low-voltage detect is disabled. The LVDE and/or LVDSE bit in SPMSC1 register is cleared.
- Flash programming/erasing is not allowed

After these conditions are met, low-power run mode can be entered by setting SPMSC2[LPR].

To re-enter standard run mode, clear the LPR bit. SPMSC2[LPRS] is a read-only status bit that can be used to determine if the regulator is in full-regulation mode or not. When LPRS is cleared, the regulator is in full-regulation mode and the MCU can run at full speed in any clock mode.

Low-power run mode can return to full regulation if any interrupt occurs by setting SPMSC2[LPWUI]. The MCG FLLs can then be set for full speed immediately in the interrupt service routine.

#### 3.6.2.1 BDM in Low-Power Run Mode

Low-power run mode cannot be entered when the MCU is in active background debug mode.

---

1. If a device is in low power run mode, a falling edge on an active BKGD/MS pin exits low power run mode, clears the LPRS bits and returns the device to normal run mode.

2. System clock gating control registers 1, 2, 3 and 4

3. FLL bypassed external low-power

## 3.7 Wait Modes

### 3.7.1 Wait Mode

Wait mode is entered by executing a STOP instruction after configuring the device as shown in [Table 3-2](#). After execution of the STOP instruction, the CPU enters a low-power state in which it is not clocked.

The V1 ColdFire core does not differentiate between stop and wait modes. Both are stop from the core's perspective. The difference between the two is at the device level. In stop mode, most peripheral clocks are shut down. In wait mode, they continue to run.

XCSR[ENBDM] must be set prior to entering wait mode if the device is required to respond to BDM commands while in wait.

In all cases when BDM is in use, or must be available, the BKGD/MS function must be enabled by clearing the PTDPF1[D6] bit. See [Section 9.7, “Pin Mux Controls,”](#) for additional details.

When an interrupt request occurs, the CPU exits wait mode and resumes with exception processing, beginning with the stacking operations leading to the interrupt service routine.

### 3.7.2 Low-Power Wait Mode (LPwait)

Low-power wait mode is entered by executing a STOP instruction while the MCU is in low-power run mode and configured as shown in [Table 3-2](#). In the low-power wait mode, the on-chip voltage regulator remains in its standby state, the power consumption is reduced to a minimum that allows most modules to maintain functionality. To reduce the power consumption further, disable the clocks to all unused peripherals by clearing the corresponding bits in the SCGC registers.

Low-power run mode restrictions also apply to low-power wait mode.

If SPMSC2[LPWUI] is set when the STOP instruction is executed, the voltage regulator returns to full regulation when wait mode is exited. The MCG FLLs can be set for full speed immediately in the interrupt service routine.

If SPMSC2[LPWUI] is cleared when the STOP instruction is executed, the device returns to low-power run mode.

Any reset exits low-power wait mode, clears SPMSC2[LPR], and returns the device to normal run mode.

#### 3.7.2.1 BDM in Low-Power Wait Mode

If a device is in low-power wait mode, a falling edge on an active BKGD/MS pin exits low-power wait mode, clears the LPR and LPRS bits in SPMSC2, and returns the device to normal run mode.

## 3.8 Stop Modes

One of three stop modes is entered upon execution of a STOP instruction when SOPT1[STOPE] is set and SOPT1[WAITE] is cleared. In stop3 mode, the bus and CPU clocks are halted. If XCSR[ENBDM] is set prior to entering stop4, only the peripheral clocks are halted. The MCG module can be configured to leave the reference clocks running. See [Chapter 6, “Multipurpose Clock Generator \(MCG\),”](#) for more information.

### NOTE

If neither the WAITE or STOPE bit is set when the CPU executes a STOP instruction, the MCU does not enter either of the stop modes. Instead, the MCU initiates an illegal opcode reset if CPUCR[IRD] is cleared or an illegal instruction exception if CPUCR[IRD] is set.

The stop modes are selected by setting the appropriate bits in the system power management status and control 2 (SPMSC2) register. [Table 3-2](#) shows the control bits that affect mode selection under various conditions. The selected mode is entered following the execution of a STOP instruction.

Most background commands are not available in stop mode. The memory-access-with-status commands do not allow memory access, but they report an error indicating that the MCU is in stop or wait mode. The BACKGROUND command can wake the MCU from stop4 and enter halt mode if XCSR[ENBDM] was set prior to entering stop. After entering halt mode, all background commands are available.

The interrupts that can wake the MPU from stop modes must be masked using the module interrupt enable functions. For more information, refer to interrupts from any of the following sources: ADC, PMC, IRQ, KBI, RTC, and SCI.

### 3.8.1 Stop2 Mode

Stop2 mode is entered by executing a STOP instruction under the conditions as shown in [Table 3-2](#).

Most of the internal circuitry of the MCU is powered off in stop2 except for the RAM and optionally the RTC. After entering stop2, all I/O pin control signals are latched so that the pins retain their states during stop2.

Exit from stop2 is performed by asserting the wakeup pin (RESET/PTC3) low.

### NOTE

RESET/PTC3 functions as an active-low wakeup input when the MCU is in stop2 if the pin is configured as an input before entering stop2. The pullup on this pin is not enabled in stop2 unless PTCPE[PTCPE3] is set.

In addition, the real-time counter (RTC) can wake the MCU from stop2, if enabled.

Upon wakeup from stop2 mode, the MCU resumes as from a power-on reset (POR):

- All module control and status registers are reset, except the power management controller (SPMSC1/2/4), RTC, and debug trace buffer. Refer to the individual module chapters for more information on which other registers are unaffected by wakeup from stop2 mode.

## Modes of Operation

- The LVD reset function is enabled and the MCU remains in the reset state if  $V_{DD}$  is below the LVD trip point (low trip point selected due to POR).
- The CPU initiates reset exception processing by fetching the vectors at 0x(00)00\_0000 and 0x(00)00\_0004.

In addition to the above, upon waking up from stop2, SPMSC2[PPDF] is set. This flag is used to direct user code to go to a stop2 recovery routine. PPDF remains set and the I/O pin states remain latched until a 1 is written to SPMSC2[PPDACK].

Wakeup from stop2 can be initiated with an RTC interrupt. Unlike most other modules on the device, the RTC is not reset as a result of exiting stop2. This implies that the RTC interrupt is asserted (although masked) upon exit from stop2.

To maintain I/O states for pins configured as general-purpose I/O, application software must save their state in RAM prior to entering stop2. During the routine called after exit from STOP2, software must restore the contents of the I/O port registers to the port registers before writing to the PPDACK bit. If the port registers are not restored from RAM, the pins switch to their reset states when PPDACK is written.

For pins that were configured as peripheral I/O, reconfigure the peripheral module that interfaces to the pin before writing to PPDACK. If the peripheral module is not enabled before writing to PPDACK, the pins are controlled by their associated port control registers when the I/O latches are opened.

### 3.8.1.1 Oscillator Considerations for Stop2

If using a low-range oscillator during stop2, reconfigure the MCGC2 register before PPDACK is written. If the low-range oscillator is active when entering stop2, it remains active in stop2 regardless of the value of MCGC2[EREFSTEN]. To disable the oscillator in stop2, switch the MCG into FBI or FEI mode before executing the STOP instruction.

The crystal oscillator cannot be used in high range in Stop2. Systems which use crystals in the high frequency range (up to 25MHz) must disable the crystal oscillator and switch to FBI or FEI mode prior to executing the STOP instruction.

### 3.8.2 Stop3 Mode

Stop3 mode is entered by executing a STOP instruction under the conditions as shown in [Table 3-2](#). The states of all of the internal registers and logic, RAM contents, and I/O pin states are maintained. The on-chip regulator is placed in standby state.

Stop3 can be exited by asserting  $\overline{\text{RESET}}$  or by an interrupt from one of the following sources: RTC, ADC, IRQ, SCI, or KBI.

If stop3 is exited by the  $\overline{\text{RESET}}$  pin, the MCU is reset and operation resumes after taking the reset vector. Exit by one of the internal interrupt sources results in the MCU taking the appropriate interrupt vector.

### 3.8.3 Stop4 Mode

Stop4 is differentiated from stop2 and stop3 in that the on-chip regulator is fully engaged.

Entry into halt mode from run mode is enabled if the XCSR[ENBDM] bit is set. This register is described in [Chapter 20, “Version 1 ColdFire Debug \(CF1\\_DEBUG\).”](#) If XCSR[ENBDM] is set when the CPU executes a STOP instruction, the system clocks to the background debug logic remain active when the MCU enters stop mode. Because of this, background debug communication remains possible. If you attempt to enter stop2 or stop3 with XCSR[ENBDM] set, the MCU enters stop4 instead (see [Table 3-2](#) for details).

Stop4 is also entered if SPMSC1[LVDE, LVDSE] are set, enabling low voltage detect when the STOP instruction is executed. The LVD may only be used when the on-chip regulator is in full regulation mode. Thus, stop3 and stop2 modes are not compatible with use of the LVD.

The LVD system is capable of generating an interrupt or a reset when the supply voltage drops below the LVD voltage.

Stop4 can be exited by asserting  $\overline{\text{RESET}}$  or by an interrupt from one of the following sources: RTC, LVD, LVW, ADC, IRQ, SCI or KBI.

## 3.9 On-Chip Peripheral Modules in Stop and Low-Power Modes

When the MCU enters any stop mode (wait not included), system clocks to the internal peripheral modules are stopped. Even in the exception case (XCSR[ENBDM] = 1), where clocks to the background debug logic continue to operate, clocks to the peripheral systems are halted to reduce power consumption. Refer to [Section 3.8.1, “Stop2 Mode,”](#) and [Section 3.8.2, “Stop3 Mode,”](#) for specific information on system behavior in stop modes.

When the MCU enters LPwait or LPrun modes, system clocks to the internal peripheral modules continue based on the settings of the clock gating control registers (SCGC1-4).

[Table 3-4](#) defines terms used in [Table 3-5](#) to describe operation of components on the device in the various low-power modes.

**Table 3-4. Abbreviations used in Table 3-5**

Voltage Regulator	Clocked <sup>1</sup>	Not Clocked
Full Regulation	FullOn	FullNoClk FullADACK <sup>2</sup>
Soft Regulation	SoftOn <sup>3</sup>	SoftNoClk Disabled SoftADACK <sup>4</sup>
Off	N/A	Off

<sup>1</sup> Subject to module enables and settings of System Clock Gating Control Registers 1 through 4 (SCGC1-4).

<sup>2</sup> ADC-specific mode where the device is fully regulated and the normal peripheral clock is stopped. The ADC can run using its internally generated asynchronous ADACK clock.

## Modes of Operation

- <sup>3</sup> Analog modules must be in their low-power mode when the device is operated in this state.
- <sup>4</sup> ADC-specific mode where the device is in soft regulation and the normal peripheral clock is stopped. The ADC can only be run using its low-power mode and internally generated asynchronous ADACK clock.

**Table 3-5. Low-Power Mode Behavior**

Peripheral	Mode					
	Stop2	Stop3	Stop4	LPwait	Wait	LPrun
<b>CF1_CORE</b>	Off	SoftNoClk	FullNoClk	SoftNoClk	FullNoClk	SoftOn
<b>RAM</b>	SoftNoClk	SoftNoClk	FullNoClk	SoftOn	FullOn	SoftOn
<b>Flash</b>	Off	SoftNoClk	FullNoClk	SoftNoClk	FullNoClk	SoftOn
<b>Port I/O Registers</b>	Off	SoftNoClk	FullNoClk	SoftOn	FullOn	SoftOn
<b>I/O Pins</b>	States Held	SoftNoClk	FullNoClk	SoftOn	FullOn	SoftOn
<b>Mini-FlexBus</b>	Pin States Held <sup>1</sup>	SoftNoClk	FullNoClk	SoftOn	FullOn	SoftOn
<b>ADC<sup>2,3</sup></b>	Off	SoftADACK (Wakeup)	FullADACK (Wakeup)	SoftOn	FullOn	SoftOn
<b>FEC</b>	Off	SoftNoClk	FullNoClk	Must be disabled	FullOn	Must be disabled
<b>BDC</b>	Off	SoftOn	On	SoftOn	FullOn	SoftOn
<b>COP</b>	Off	SoftNoClk	FullNoClk	SoftOn	FullOn	SoftOn
<b>Crystal Oscillator</b>	RANGE=0 HGO=0	RANGE=0 HGO=0	All Modes	RANGE=0 HGO=0	All Modes	RANGE=0 HGO=0
<b>MCG</b>	Off	Stop or BLPE <sup>4</sup>	Stop or any mode	BLPE	Any mode	BLPE
<b>IICx</b>	Off	SoftNoClk	FullNoClk	SoftOn	FullOn	SoftOn
<b>IRQ</b>	Off (Wakeup via POR) <sup>5</sup>	SoftNoClk (Wakeup)	FullNoClk (Wakeup)	SoftOn	FullOn	SoftOn
<b>KBIx</b>	Off	SoftNoClk (Wakeup)	FullNoClk (Wakeup)	SoftOn	FullOn	SoftOn
<b>LVD/LVW</b>	Off	Disabled	On (Wakeup)	Disabled	FullOn	Disabled
<b>RTC</b>	Soft Regulation, LPOCLK if enabled (Wakeup via POR)	SoftOn (Wakeup)	Full Regulation, LPOCLK (Wakeup)	SoftOn	FullOn	SoftOn
<b>SCIx</b>	Off	SoftNoClk	FullNoClk	SoftOn	FullOn	SoftOn

**Table 3-5. Low-Power Mode Behavior (continued)**

Peripheral	Mode					
	Stop2	Stop3	Stop4	LPwait	Wait	LPrun
<b>SPIx</b>	Off	SoftNoClk	FullNoClk	SoftOn	FullOn	SoftOn
<b>TPMx</b>	Off	SoftNoClk	FullNoClk	SoftOn	FullOn	SoftOn
<b>Voltage Regulator / PMC</b>	Partial Shutdown. 1 kHz osc if enabled	Soft Regulation. 1 kHz osc if enabled	Full Regulation 1 kHz osc on	SoftOn 1 kHz osc on	FullOn 1 kHz osc on	SoftOn 1 kHz osc on

<sup>1</sup> This behavior is due to operation of the I/O cells in conjunction with the PMC. The Mini-FlexBus does not save state during the transitions through stop2.

<sup>2</sup> LP mode for the ADC is invoked by setting ADLPC. ADACK is selected by the ADCCFG[ADICLK] field in the ADC. See [Chapter 15, “Analog-to-Digital Converter \(ADC12\),”](#) for details.

<sup>3</sup> LVD must be enabled to run in stop if converting the bandgap channel.

<sup>4</sup> BLPE refers to the MCG bypassed external low-power state. See [Chapter 6, “Multipurpose Clock Generator \(MCG\),”](#) for more details.

<sup>5</sup> The RESET/PTC3 pin also has a direct connection to the on-chip regulator wakeup input. Asserting this pin low while in stop2 triggers the PMC to wakeup. As a result, the device undergoes a power-on-reset sequence.

# Chapter 4

## Memory

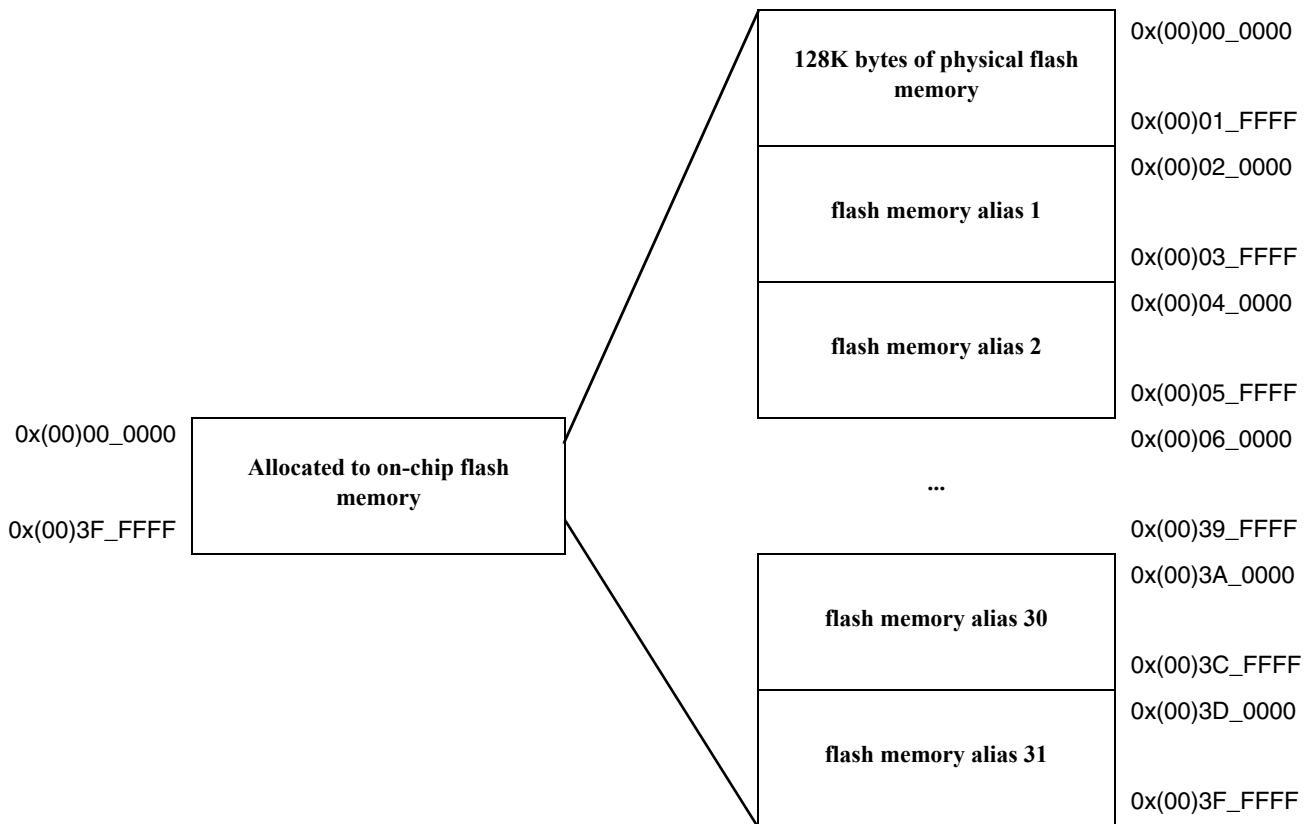
### 4.1 MCF51CN128 Series Memory Map

80-pin versions of the device allow flash and memory to be supplemented with off-chip storage via the Mini-FlexBus, providing a continuous memory range from 0x(00)00 0000 to 0x(00)C0 0000.

Address Range	V1 ColdFire Memory Usage
0x(00)00_0000	Allocated to on-chip flash memory
0x(00)3F_FFFF	
0x(00)40_0000	Available for off-chip expansion
0x(00)7F_FFFF	
0x(00)80_0000	Allocated to on-chip RAM memory
0x(00)9F_FFFF	
0x(00)A0_0000	Available for off-chip expansion
0x(00)BF_FFFF	
0x(00)C0_0000	ColdFire Rapid GPIO
0x(00)C0_000F	
0x(00)C0_0010	Unimplemented
0x(FF)FF_7FFF	
0x(FF)FF_8000	Slave Peripherals
0x(FF)FF_FFFF	

Figure 4-1. Generic V1 ColdFire Memory Map

[Figure 4-1](#) is the generic, high level, memory map applicable to the MCF51CN128 family. Devices (including the MCF51CN128) which include off-chip expansion capability alias smaller memories across the range shown as shown in [Figure 4-2](#).<sup>1</sup>



**Figure 4-2. Flash Memory Aliasing**

The figure shows that MCF51CN128 includes 128K bytes of flash memory. This memory is 32 times beginning at 0x(00)00\_0000, but is also aliased at 0x(00)02\_0000, 0X(00)04\_0000, 0x(00)06\_0000 and so on up through 0x(00)3D\_0000.

The region from 0x(00)40\_0000 through 0x(00)7F\_FFFF can be used by the Mini-FlexBus for off-chip expansion, allowing for natural code expansion from flash to off-chip memory if required.

The 2 MB region at 0x(00)80\_0000 is allocated for on-chip RAM. Only 24 KB of that is physically implemented on the MCF51CN128. RAM address decoding is aliased every 32 KB across the 2MB region from 0x(00)80\_0000 to 0x(00)9F\_FFFF.

Regions within the memory map are subject to restrictions with regard to the types of CPU accesses allowed. These are outlined in [Table 4-1](#). Non-supported access types terminate the bus cycle with an error (and typically generate a system reset in response to the error termination).

1. This is not necessarily true of parts with no external expansion port.

**Table 4-1. CPU Access Type Allowed by Region**

Base Address	Region	Read			Write		
		Byte	Word	Long	Byte	Word	Long
0x(00)00_0000	Flash	x	x	x	—	—	x
0x(00)80_0000	RAM	x	x	x	x	x	x
0x(00)C0_0000	Rapid GPIO	x	x	x	x	x	x
0x(FF)FF_8000	8-bit Peripherals <sup>1</sup>	x	x	x	x	x	x
0x(FF)FF_E000	Fast Ethernet Controller	—	—	x	—	—	x
0x(FF)FF_E800	Mini-FlexBus	x	x	x	x	x	x

<sup>1</sup> Allowed access types are peripheral specific. The peripheral bus bridge serializes 16 and 32-bit accesses into multiple 8-bit accesses. When using 8-bit peripherals, ensure that all accesses are properly aligned and only desired 8-bit locations are accessed.

Consistent with past ColdFire devices, flash configuration data is located at 0x(00)00\_0400. The slave peripherals section of the memory map is further broken down as shown in [Table 4-2](#).

**Table 4-2. High Level Peripheral Memory Map**

Peripheral	Description	Instance Name	BaseAddress
RGPIO	Rapid General Purpose I/O	RGPIO	0x(00)C0_0000
Port I/O Module	General Purpose I/O	PTA	0x(FF)FF_8000
Port Control Module	Port I/O Control Module	PTA	0x(FF)FF_8008
Port I/O Module	General Purpose I/O	PTB	0x(FF)FF_8010
Port Control Module	Port I/O Control Module	PTB	0x(FF)FF_8018
Port I/O Module	General Purpose I/O	PTC	0x(FF)FF_8020
Port Control Module	Port I/O Control Module	PTC	0x(FF)FF_8028
Port I/O Module	General Purpose I/O	PTD	0x(FF)FF_8030
Port Control Module	Port I/O Control Module	PTD	0x(FF)FF_8038
Port I/O Module	General Purpose I/O	PTE	0x(FF)FF_8040
Port Control Module	Port I/O Control Module	PTE	0x(FF)FF_8048
KBI	Keyboard Interrupt Module	KBI2	0x(FF)FF_804C
Port I/O Module	General Purpose I/O	PTF	0x(FF)FF_8050
Port Control Module	Port I/O Control Module	PTF	0x(FF)FF_8058
Port I/O Module	General Purpose I/O	PTG	0x(FF)FF_8060

**Table 4-2. High Level Peripheral Memory Map (continued)**

Peripheral	Description	Instance Name	BaseAddress
Port Control Module	Port I/O Control Module	PTG	0x(FF)FF_8068
KBI	Keyboard Interrupt Module	KBI1	0x(FF)FF_806C
Port I/O Module	General Purpose I/O	PTH	0x(FF)FF_8070
Port Control Module	Port I/O Control Module	PTH	0x(FF)FF_8078
Port I/O Module	General Purpose I/O	PTJ	0x(FF)FF_8080
Port Control Module	Port I/O Control Module	PTJ	0x(FF)FF_8088
Port Mux Controls	Port Mux Controls	MC	0x(FF)FF_80C0
IRQ	External Interrupt Module	IRQ	0x(FF)FF_80E0
SIM	System Integration Module	SIM	0x(FF)FF_8100
PMC	Power Management Controller	PMC	0x(FF)FF_8120
ADC	12-bit Successive Approximation Analog-to-Digital Converter	ADC	0x(FF)FF_8140
SCI	Serial Communications Interface	SCI1	0x(FF)FF_8160
SCI	Serial Communications Interface	SCI2	0x(FF)FF_8180
SCI	Serial Communications Interface	SCI3	0x(FF)FF_81A0
SPI	Serial Peripheral Interface	SPI1	0x(FF)FF_81C0
SPI	Serial Peripheral Interface	SPI2	0x(FF)FF_81E0
IIC	Inter-Integrated IC	IIC1	0x(FF)FF_8200
IIC	Inter-Integrated IC	IIC2	0x(FF)FF_8220
MCG	Multipurpose Clock Generator	MCG	0x(FF)FF_8240
3-channel TPM	3-channel Timer / PWM Module	TPM1	0x(FF)FF_8260
3-channel TPM	3-channel Timer / PWM Module	TPM2	0x(FF)FF_8280
MTIM8	8-Bit Modulo Timer	MTIM1	0x(FF)FF_82A0
RTC	Real Time Counter	RTC	0x(FF)FF_82C0
FTSR	Flash Wrapper	FTSR	0x(FF)FF_82E0
MTIM8	8-Bit Modulo Timer	MTIM2	0x(FF)FF_8300
FEC	Fast Ethernet Controller	FEC	0x(FF)FF_E000
Mini-FlexBus	External Memory Interface	MB	0x(FF)FF_E800
INTC	V1 ColdFire Interrupt Controller	INTC	0x(FF)FF_FFC0

The section of memory at 0x(00)C0\_0000 is used by the ColdFire Rapid GPIO module. See [Table 4-6](#) for the Rapid GPIO memory map and [Chapter 10, “Rapid GPIO \(RGPIO\),”](#) for further details on the module.

The MCF51CN128 series microcontrollers use an 8-bit peripheral bus. The bus bridge from the ColdFire system bus to the peripheral bus is capable of serializing 16-bit accesses into two 8-bit accesses and 32-bit access into four 8-bit accesses. This can be used to speed access to properly aligned peripheral registers. However, not all peripheral registers are aligned to take advantage of this feature.

CPU accesses to those parts of the memory map marked as reserved in [Figure 4-1](#) result in an illegal address reset if CPUCR[ARD] = 0 or an address error exception if CPUCR[ARD] = 1.

The lower 32 KB of flash memory and slave peripherals section of the memory map are most efficiently accessed using the ColdFire absolute short addressing mode. RAM is most efficiently accessed using the A5-relative addressing mode (address register indirect with displacement mode).

## 4.2 Detailed Register Addresses and Bit Assignments

The ColdFire interrupt controller module is mapped in the peripheral space and occupies a 64-byte space at the upper end of memory. Accordingly, its address decode is defined as

0x(FF)FF\_FFC0–0x(FF)FF\_FFFF. This 64-byte space includes the program-visible interrupt controller registers and the space used for interrupt acknowledge (IACK) cycles.

There is a nonvolatile register area consisting of a block of 16 bytes in flash memory at 0x(00)00\_0400–0x(00)00\_040F. Nonvolatile register locations include:

- NVPROT and NVOPT are loaded into working registers at reset
- An 8-byte backdoor comparison key that optionally allows a user to gain controlled access to secure memory

Because the nonvolatile register locations are flash memory, they must be erased and programmed like other flash memory locations.

[Table 4-3](#) is a summary of all user-accessible peripheral registers and control bits.

- Cells that are not associated with named bits are shaded.
- Shaded cell with zeroes indicates this unused bit always reads as a zero.
- Shaded cells with dashes indicate unused or reserved bit locations that could read as ones or zeroes. When writing to these bits, write a zero unless otherwise specified.
- Reserved registers may read indeterminate values and must not be written.

**Table 4-3. Detailed Peripheral Memory Map**

Address	Peripheral	Register	Bit 15/7	14/6	13/5	12/4	11/3	10/2	9/1	Bit 8/0
0x(00)C0_0000	GPIO	GPIO_DIR				DIR[15:8] (Read/Write)				
						DIR[7:0] (Read/Write)				
0x(00)C0_0002	GPIO	GPIO_DATA				DATA[15:8] (Read/Write)				
						DATA[7:0] (Read/Write)				
0x(00)C0_0004	GPIO	GPIO_ENB				ENB[15:8] (Read/Write)				
						ENB[7:0] (Read/Write)				
0x(00)C0_0006	GPIO	GPIO_CLR				CLR[15:8] (Write only)				

**Table 4-3. Detailed Peripheral Memory Map (continued)**

			CLR[7:0] (Write only)							
0x(00)C0_0006	GPIO	RESERVED	DATA[15:8] (Read only)							
			DATA[7:0] (Read only)							
0x(00)C0_0008	GPIO	RESERVED	DIR[15:8] (Read only)							
			DIR[7:0] (Read only)							
0x(00)C0_000A	GPIO	GPIO_SET	SET[15:8] (Write only)							
			SET[7:0] (Write only)							
0x(00)C0_000A	GPIO	RESERVED	DATA[15:8] (Read only)							
			DATA[7:0] (Read only)							
0x(00)C0_000C	GPIO	RESERVED	DIR[15:8] (Read only)							
			DIR[7:0] (Read only)							
0x(00)C0_000E	GPIO	GPIO_TOG	TOG[15:8] (Write only)							
			TOG[17:0] (Write only)							
0x(00)C0_000E	GPIO	RESERVED	DATA[15:8] (Read only)							
			DATA[7:0] (Read only)							
Address	Peripheral	Register	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_8000	PTA	PTAD	PTAD7	PTAD6	PTAD5	PTAD4	PTAD3	PTAD2	PTAD1	PTAD0
0x(FF)FF_8001	PTA	PTADD	PTADD7	PTADD6	PTADD5	PTADD4	PTADD3	PTADD2	PTADD1	PTADD0
Address	Peripheral	Register	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_8008	PTA	PTAPE	PTAPE7	PTAPE6	PTAPE5	PTAPE4	PTAPE3	PTAPE2	PTAPE1	PTAPE0
0x(FF)FF_8009	PTA	PTASE	PTASE7	PTASE6	PTASE5	PTASE4	PTASE3	PTASE2	PTASE1	PTASE0
0x(FF)FF_800A	PTA	PTADS	PTADS7	PTADS6	PTADS5	PTADS4	PTADS3	PTADS2	PTADS1	PTADS0
0x(FF)FF_800B	PTA	PTAIFE	PTAIFE7	PTAIFE6	PTAIFE5	PTAIFE4	PTAIFE3	PTAIFE2	PTAIFE1	PTAIFE0
Address	Peripheral	Register	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_8010	PTB	PTBD	PTBD7	PTBD6	PTBD5	PTBD4	PTBD3	PTBD2	PTBD1	PTBD0
0x(FF)FF_8011	PTB	PTBDD	PTBDD7	PTBDD6	PTBDD5	PTBDD4	PTBDD3	PTBDD2	PTBDD1	PTBDD0
Address	Peripheral	Register	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_8018	PTB	PTBPE	PTBPE7	PTBPE6	PTBPE5	PTBPE4	PTBPE3	PTBPE2	PTBPE1	PTBPE0
0x(FF)FF_8019	PTB	PTBSE	PTBSE7	PTBSE6	PTBSE5	PTBSE4	PTBSE3	PTBSE2	PTBSE1	PTBSE0
0x(FF)FF_801A	PTB	PTBDS	PTBDS7	PTBDS6	PTBDS5	PTBDS4	PTBDS3	PTBDS2	PTBDS1	PTBDS0
0x(FF)FF_801B	PTB	PTBIFE	PTBIFE7	PTBIFE6	PTBIFE5	PTBIFE4	PTBIFE3	PTBIFE2	PTBIFE1	PTBIFE0
Address	Peripheral	Register	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_8020	PTC	PTCD	PTCD7	PTCD6	PTCD5	PTCD4	PTCD3	PTCD2	PTCD1	PTCD0
0x(FF)FF_8021	PTC	PTCDD	PTCDD7	PTCDD6	PTCDD5	PTCDD4	PTCDD3	PTCDD2	PTCDD1	PTCDD0
Address	Peripheral	Register	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_8028	PTC	PTCPE	PTCPE7	PTCPE6	PTCPE5	PTCPE4	PTCPE3	PTCPE2	PTCPE1	PTCPE0
0x(FF)FF_8029	PTC	PTCSE	PTCSE7	PTCSE6	PTCSE5	PTCSE4	PTCSE3	PTCSE2	PTCSE1	PTCSE0

**Table 4-3. Detailed Peripheral Memory Map (continued)**

0x(FF)FF_802A	PTC	PTCDS	PTCDS7	PTCDS6	PTCDS5	PTCDS4	PTCDS3	PTCDS2	PTCDS1	PTCDS0
0x(FF)FF_802B	PTC	PTCIFE	PTCIFE7	PTCIFE6	PTCIFE5	PTCIFE4	PTCIFE3	PTCIFE2	PTCIFE1	PTCIFE0
Address	Peripheral	Register	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_8030	PTD	PTDD	PTDD7	PTDD6	PTDD5	PTDD4	PTDD3	PTDD2	PTDD1	PTDD0
0x(FF)FF_8031	PTD	PTDDD	PTDDD7	PTDDD6	PTDDD5	PTDDD4	PTDDD3	PTDDD2	PTDDD1	PTDDD0
Address	Peripheral	Register	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_8038	PTD	PTDPE	PTDPE7	PTDPE6	PTDPE5	PTDPE4	PTDPE3	PTDPE2	PTDPE1	PTDPE0
0x(FF)FF_8039	PTD	PTDSE	PTDSE7	PTDSE6	PTDSE5	PTDSE4	PTDSE3	PTDSE2	PTDSE1	PTDSE0
0x(FF)FF_803A	PTD	PTDDS	PTDDS7	PTDDS6	PTDDS5	PTDDS4	PTDDS3	PTDDS2	PTDDS1	PTDDS0
0x(FF)FF_803B	PTD	PTDIFE	PTDIFE7	PTDIFE6	PTDIFE5	PTDIFE4	PTDIFE3	PTDIFE2	PTDIFE1	PTDIFE0
Address	Peripheral	Register	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_8040	PTE	PTED	PTED7	PTED6	PTED5	PTED4	PTED3	PTED2	PTED1	PTED0
0x(FF)FF_8041	PTE	PTEDD	PTEDD7	PTEDD6	PTEDD5	PTEDD4	PTEDD3	PTEDD2	PTEDD1	PTEDD0
Address	Peripheral	Register	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_8048	PTE	PTEPE	PTEPE7	PTEPE6	PTEPE5	PTEPE4	PTEPE3	PTEPE2	PTEPE1	PTEPE0
0x(FF)FF_8049	PTE	PTESE	PTESE7	PTESE6	PTESE5	PTESE4	PTESE3	PTESE2	PTESE1	PTESE0
0x(FF)FF_804A	PTE	PTEDS	PTEDS7	PTEDS6	PTEDS5	PTEDS4	PTEDS3	PTEDS2	PTEDS1	PTEDS0
0x(FF)FF_804B	PTE	PTEIFE	PTEIFE7	PTEIFE6	PTEIFE5	PTEIFE4	PTEIFE3	PTEIFE2	PTEIFE1	PTEIFE0
Address	Peripheral	Register	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_804C	KBI2	KBI2SC	0	0	0	0	KBF	KBACK	KBIE	KBIMOD
0x(FF)FF_804D	KBI2	KBI2PE	KBIPE7	KBIPE6	KBIPE5	KBIPE4	KBIPE3	KBIPE2	KBIPE1	KBIPE0
0x(FF)FF_804E	KBI2	KBI2ES	KBEDG7	KBEDG6	KBEDG5	KBEDG4	KBEDG3	KBEDG2	KBEDG1	KBEDG0
Address	Peripheral	Register	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_8050	PTF	PTFD	PTFD7	PTFD6	PTFD5	PTFD4	PTFD3	PTFD2	PTFD1	PTFD0
0x(FF)FF_8051	PTF	PTFDD	PTFDD7	PTFDD6	PTFDD5	PTFDD4	PTFDD3	PTFDD2	PTFDD1	PTFDD0
Address	Peripheral	Register	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_8058	PTF	PTFPE	PTFPE7	PTFPE6	PTFPE5	PTFPE4	PTFPE3	PTFPE2	PTFPE1	PTFPE0
0x(FF)FF_8059	PTF	PTFSE	PTFSE7	PTFSE6	PTFSE5	PTFSE4	PTFSE3	PTFSE2	PTFSE1	PTFSE0
0x(FF)FF_805A	PTF	PTFDS	PTFDS7	PTFDS6	PTFDS5	PTFDS4	PTFDS3	PTFDS2	PTFDS1	PTFDS0
0x(FF)FF_805B	PTF	PTFIFE	PTFIFE7	PTFIFE6	PTFIFE5	PTFIFE4	PTFIFE3	PTFIFE2	PTFIFE1	PTFIFE0
Address	Peripheral	Register	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_8060	PTG	PTGD	PTGD7	PTGD6	PTGD5	PTGD4	PTGD3	PTGD2	PTGD1	PTGD0
0x(FF)FF_8061	PTG	PTGDD	PTGDD7	PTGDD6	PTGDD5	PTGDD4	PTGDD3	PTGDD2	PTGDD1	PTGDD0
Address	Peripheral	Register	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_8068	PTG	PTGPE	PTGPE7	PTGPE6	PTGPE5	PTGPE4	PTGPE3	PTGPE2	PTGPE1	PTGPE0
0x(FF)FF_8069	PTG	PTGSE	PTGSE7	PTGSE6	PTGSE5	PTGSE4	PTGSE3	PTGSE2	PTGSE1	PTGSE0

**Table 4-3. Detailed Peripheral Memory Map (continued)**

0x(FF)FF_806A	PTG	PTGDS	PTGDS7	PTGDS6	PTGDS5	PTGDS4	PTGDS3	PTGDS2	PTGDS1	PTGDS0
0x(FF)FF_806B	PTG	PTGIFE	PTGIFE7	PTGIFE6	PTGIFE5	PTGIFE4	PTGIFE3	PTGIFE2	PTGIFE1	PTGIFE0
Address	Peripheral	Register	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_806C	KBI1	KBI1SC	0	0	0	0	KBF	KBACK	KBIE	KBIMOD
0x(FF)FF_806D	KBI1	KBI1PE	KBIPE7	KBIPE6	KBIPE5	KBIPE4	KBIPE3	KBIPE2	KBIPE1	KBIPE0
0x(FF)FF_806E	KBI1	KBI1ES	KBEDG7	KBEDG6	KBEDG5	KBEDG4	KBEDG3	KBEDG2	KBEDG1	KBEDG0
Address	Peripheral	Register	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_8070	PTH	PTHHD	PTHHD7	PTHHD6	PTHHD5	PTHHD4	PTHHD3	PTHHD2	PTHHD1	PTHHD0
0x(FF)FF_8071	PTH	PTHDD	PTHDD7	PTHDD6	PTHDD5	PTHDD4	PTHDD3	PTHDD2	PTHDD1	PTHDD0
Address	Peripheral	Register	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_8078	PTH	PTHPE	PTHPE7	PTHPE6	PTHPE5	PTHPE4	PTHPE3	PTHPE2	PTHPE1	PTHPE0
0x(FF)FF_8079	PTH	PTHSE	PTHSE7	PTHSE6	PTHSE5	PTHSE4	PTHSE3	PTHSE2	PTHSE1	PTHSE0
0x(FF)FF_807A	PTH	PTHDS	PTHDS7	PTHDS6	PTHDS5	PTHDS4	PTHDS3	PTHDS2	PTHDS1	PTHDS0
0x(FF)FF_807B	PTH	PTHIFE	PTHIFE7	PTHIFE6	PTHIFE5	PTHIFE4	PTHIFE3	PTHIFE2	PTHIFE1	PTHIFE0
Address	Peripheral	Register	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_8080	PTJ	PTJD	PTJD7	PTJD6	PTJD5	PTJD4	PTJD3	PTJD2	PTJD1	PTJD0
0x(FF)FF_8081	PTJ	PTJDD	PTJDD7	PTJDD6	PTJDD5	PTJDD4	PTJDD3	PTJDD2	PTJDD1	PTJDD0
Address	Peripheral	Register	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_8088	PTJ	PTJPE	PTJPE7	PTJPE6	PTJPE5	PTJPE4	PTJPE3	PTJPE2	PTJPE1	PTJPE0
0x(FF)FF_8089	PTJ	PTJSE	PTJSE7	PTJSE6	PTJSE5	PTJSE4	PTJSE3	PTJSE2	PTJSE1	PTJSE0
0x(FF)FF_808A	PTJ	PTJDS	PTJDS7	PTJDS6	PTJDS5	PTJDS4	PTJDS3	PTJDS2	PTJDS1	PTJDS0
0x(FF)FF_808B	PTJ	PTJIFE	PTJIFE7	PTJIFE6	PTJIFE5	PTJIFE4	PTJIFE3	PTJIFE2	PTJIFE1	PTJIFE0
Address	Peripheral	Register	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_80C0	MC	PTAPF1	A7		A6		A5		A4	
0x(FF)FF_80C1	MC	PTAPF2	A3		A2		A1		A0	
0x(FF)FF_80C2	MC	PTBPF1	B7		B6		B5		B4	
0x(FF)FF_80C3	MC	PTBPF2	B3		B2		B1		B0	
0x(FF)FF_80C4	MC	PTCPF1	C7		C6		C5		C4	
0x(FF)FF_80C5	MC	PTCPF2	C3		C2		C1		C0	
0x(FF)FF_80C6	MC	PTDPF1	D7		D6		D5		D4	
0x(FF)FF_80C7	MC	PTDPF2	D3		D2		D1		D0	
0x(FF)FF_80C8	MC	PTEPF1	E7		E6		E5		E4	
0x(FF)FF_80C9	MC	PTEPF2	E3		E2		E1		E0	
0x(FF)FF_80CA	MC	PTFPF1	F7		F6		F5		F4	
0x(FF)FF_80CB	MC	PTFPF2	F3		F2		F1		F0	
0x(FF)FF_80CC	MC	PTGPF1	G7		G6		G5		G4	
0x(FF)FF_80CD	MC	PTGPF2	G3		G2		G1		G0	

**Table 4-3. Detailed Peripheral Memory Map (continued)**

0x(FF)FF_80CE	MC	PTHPF1	H7		H6		H5		H4			
0x(FF)FF_80CF	MC	PTHPF2	H3		H2		H1		H0			
0x(FF)FF_80D0	MC	PTJPF1	0	0	0	0	J5		J4			
0x(FF)FF_80D1	MC	PTJPF2	J3		J2		J1		J0			
Address	Peripheral	Register	Bit 7	6	5	4	3	2	1	Bit 0		
0x(FF)FF_80E0	IRQ	IRQSC	0	IRQPDD	IRQEDG	IRQPE	IRQF	IRQACK	IRQIE	IRQMOD		
Address	Peripheral	Register	Bit 7	6	5	4	3	2	1	Bit 0		
0x(FF)FF_8100	SIM	SRS	POR	PIN	COP	ILOP	ILAD	LOC	LVD	0		
0x(FF)FF_8101	SIM	SOPT1	0	SL	STOPE	WAITE	COPT		COPCLKS	COPW		
0x(FF)FF_8102	SIM	SOPT2	TOV	FC		PMC_LVD_TRIM						
0x(FF)FF_8103	SIM	SOPT3	0	0	0	CS			PCS			
0x(FF)FF_8104	SIM	RESERVED	—				—	—	—			
0x(FF)FF_8105	SIM	RESERVED	—				—	—	—	—		
0x(FF)FF_8106	SIM	SDIDH	REV				ID11	ID10	ID9	ID8		
0x(FF)FF_8107	SIM	SDIDL	ID7	ID6	ID5	ID4	ID3	ID2	ID1	ID0		
0x(FF)FF_8108	SIM	SCGC1	MTIM	TPM2	TPM1	ADC	IIC2	IIC1	SCI2	SCI1		
0x(FF)FF_8109	SIM	SCGC2	SCI3	FTSR	IRQ	KBI2	KBI1	RTC	SPI2	SPI1		
0x(FF)FF_810A	SIM	SCGC3	PTH	PTG	PTF	PTE	PTD	PTC	PTB	PTA		
0x(FF)FF_810B	SIM	SCGC4	0	0	0	MTIM2	MC	MB	FEC	PTJ		
0x(FF)FF_810C	SIM	SIMIPS	0	0	0	0	TPM2	TPM1	MTIM2	MTIM1		
Address	Peripheral	Register	Bit 7	6	5	4	3	2	1	Bit 0		
0x(FF)FF_8120	PMC	SPMSC1	LVDF	LVDACK	LVDIE	LVDRE	LVDSE	LVDE	0	BGBE		
0x(FF)FF_8121	PMC	SPMSC2	LPR	LPRS	LPWUI	0	PPDF	PPDACK	PPDE	PPDC		
0x(FF)FF_8122	PMC	RESERVED	—	—	—	—	—	—	—	—		
0x(FF)FF_8123	PMC	SPMSC3	LVWF	LVWACK	LVDV	LVVV	LVWIE	0	0	0		
Address	Peripheral	Register	Bit 7	6	5	4	3	2	1	Bit 0		
0x(FF)FF_8140	ADC	ADCSC1	COCO	AIEN	ADCO	ADCH						
0x(FF)FF_8141	ADC	ADCSC2	ADACT	ADTRG	0	0	0	0	REFSEL			
0x(FF)FF_8142	ADC	ADCRH	0	0	0	0	ADR11	ADR10	ADR9	ADR8		
0x(FF)FF_8143	ADC	ADCRL	ADR7	ADR6	ADR5	ADR4	ADR3	ADR2	ADR1	ADR0		
0x(FF)FF_8144	ADC	ADCCVH	0	0	0	0	ADCV11	ADCV10	ADCV9	ADCV8		
0x(FF)FF_8145	ADC	ADCCVL	ADCV7	ADCV6	ADCV5	ADCV4	ADCV3	ADCV2	ADCV1	ADCV0		
0x(FF)FF_8146	ADC	ADCCFG	ADLPC	ADIV		ADLSMP	MODE		ADICLK			
0x(FF)FF_8147	ADC	RESERVED	—	—	—	—	—	—	—	—		
0x(FF)FF_8149	ADC	RESERVED	—	—	—	—	—	—	—	—		
Address	Peripheral	Register	Bit 7	6	5	4	3	2	1	Bit 0		

**Table 4-3. Detailed Peripheral Memory Map (continued)**

0x(FF)FF_8160	SCI1	SCI1BDH	LBK DIE	RXEDGI E	0	SBR12	SBR11	SBR10	SBR9	SBR8
0x(FF)FF_8161	SCI1	SCI1BDL	SBR7	SBR6	SBR5	SBR4	SBR3	SBR2	SBR1	SBR0
0x(FF)FF_8162	SCI1	SCI1C1	LOOPS	SCISWAI	RSRC	M	WAKE	ILT	PE	PT
0x(FF)FF_8163	SCI1	SCI1C2	TIE	TCIE	RIE	ILIE	TE	RE	RWU	SBK
0x(FF)FF_8164	SCI1	SCI1S1	TDRE	TC	RDRF	IDLE	OR	NF	FE	PF
0x(FF)FF_8165	SCI1	SCI1S2	LBKDIF	RXEDGI F	0	RXINV	RWUID	BRK13	LBKDE	RAF
0x(FF)FF_8166	SCI1	SCI1C3	R8	T8	TXDIR	TXINV	ORIE	NEIE	FEIE	PEIE
0x(FF)FF_8167	SCI1	SCI1D	DATA							
Address	Peripheral	Register	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_8180	SCI2	SCI2BDH	LBK DIE	RXEDGI E	0	SBR12	SBR11	SBR10	SBR9	SBR8
0x(FF)FF_8181	SCI2	SCI2BDL	SBR7	SBR6	SBR5	SBR4	SBR3	SBR2	SBR1	SBR0
0x(FF)FF_8182	SCI2	SCI2C1	LOOPS	SCISWAI	RSRC	M	WAKE	ILT	PE	PT
0x(FF)FF_8183	SCI2	SCI2C2	TIE	TCIE	RIE	ILIE	TE	RE	RWU	SBK
0x(FF)FF_8184	SCI2	SCI2S1	TDRE	TC	RDRF	IDLE	OR	NF	FE	PF
0x(FF)FF_8185	SCI2	SCI2S2	LBKDIF	RXEDGI F	0	RXINV	RWUID	BRK13	LBKDE	RAF
0x(FF)FF_8186	SCI2	SCI2C3	R8	T8	TXDIR	TXINV	ORIE	NEIE	FEIE	PEIE
0x(FF)FF_8187	SCI2	SCI2D	DATA							
Address	Peripheral	Register	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_81A0	SCI3	SCI3BDH	LBK DIE	RXEDGI E	0	SBR12	SBR11	SBR10	SBR9	SBR8
0x(FF)FF_81A1	SCI3	SCI3BDL	SBR7	SBR6	SBR5	SBR4	SBR3	SBR2	SBR1	SBR0
0x(FF)FF_81A2	SCI3	SCI3C1	LOOPS	SCISWAI	RSRC	M	WAKE	ILT	PE	PT
0x(FF)FF_81A3	SCI3	SCI3C2	TIE	TCIE	RIE	ILIE	TE	RE	RWU	SBK
0x(FF)FF_81A4	SCI3	SCI3S1	TDRE	TC	RDRF	IDLE	OR	NF	FE	PF
0x(FF)FF_81A5	SCI3	SCI3S2	LBKDIF	RXEDGI F	0	RXINV	RWUID	BRK13	LBKDE	RAF
0x(FF)FF_81A6	SCI3	SCI3C3	R8	T8	TXDIR	TXINV	ORIE	NEIE	FEIE	PEIE
0x(FF)FF_81A7	SCI3	SCI3D	DATA							
Address	Peripheral	Register	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_81C0	SPI1	SPI1C1	SPIE	SPE	SPTIE	MSTR	CPOL	CPHA	SSOE	LSBFE
0x(FF)FF_81C1	SPI1	SPI1C2	0	0	0	MODFEN	BIDIROE	0	SPISWAI	SPC0
0x(FF)FF_81C2	SPI1	SPI1BR	0	SPPR			0	SPR		
0x(FF)FF_81C3	SPI1	SPI1S	SPRF	0	SPTEF	MODF	0	0	0	0
0x(FF)FF_81C4	SPI1	RESERVED	—							
0x(FF)FF_81C5	SPI1	SPI1D	DATA							
Address	Peripheral	Register	Bit 7	6	5	4	3	2	1	Bit 0

**Table 4-3. Detailed Peripheral Memory Map (continued)**

0x(FF)FF_81E0	SPI2	SPI2C1	SPIE	SPE	SPTIE	MSTR	CPOL	CPHA	SSOE	LSBFE
0x(FF)FF_81E1	SPI2	SPI2C2	0	0	0	MODFEN	BIDIROE	0	SPISWAI	SPC0
0x(FF)FF_81E2	SPI2	SPI2BR	0		SPPR		0		SPR	
0x(FF)FF_81E3	SPI2	SPI2S	SPRF	0	SPTEF	MODF	0	0	0	0
0x(FF)FF_81E4	SPI2	RESERVED					—			
0x(FF)FF_81E5	SPI2	SPI2D					DATA			
Address	Peripheral	Register	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_8200	IIC1	IIC1A1	AD7	AD6	AD5	AD4	AD3	AD2	AD1	0
0x(FF)FF_8201	IIC1	IIC1F	MULT				ICR			
0x(FF)FF_8202	IIC1	IIC1C1	IICEN	IICIE	MST	TX	TXAK	RSTA	0	0
0x(FF)FF_8203	IIC1	IIC1S	TCF	IAAS	BUSY	ARBL	0	SRW	IICIF	RXAK
0x(FF)FF_8204	IIC1	IIC1D					DATA			
0x(FF)FF_8205	IIC1	IIC1C2	GCAEN	ADEXT	0	0	0	AD10	AD9	AD8
0x(FF)FF_8206	IIC1	IIC1SMB	FACK	ALERTE_N	SIICAEN	TCKSEL	SLTF	SHTF	0	0
0x(FF)FF_8207	IIC1	IIC1A2	SAD7	SAD6	SAD5	SAD4	SAD3	SAD2	SAD1	0
0x(FF)FF_8208	IIC1	IIC1SLTH	SSLT15	SSLT14	SSLT13	SSLT12	SSLT11	SSLT10	SSLT9	SSLT8
0x(FF)FF_8209	IIC1	IIC1SLTL	SSLT7	SSLT6	SSLT5	SSLT4	SSLT3	SSLT2	SSLT1	SSLT0
0x(FF)FF_820A	IIC1	IC1FLT	0	0	0	0	FLT3	FLT2	FLT1	FLT0
Address	Peripheral	Register	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_8220	IIC2	IIC2A1	AD7	AD6	AD5	AD4	AD3	AD2	AD1	0
0x(FF)FF_8221	IIC2	IIC2F	MULT				ICR			
0x(FF)FF_8222	IIC2	IIC2C1	IICEN	IICIE	MST	TX	TXAK	RSTA	0	0
0x(FF)FF_8223	IIC2	IIC2S	TCF	IAAS	BUSY	ARBL	0	SRW	IICIF	RXAK
0x(FF)FF_8224	IIC2	IIC2D					DATA			
0x(FF)FF_8225	IIC2	IIC2C2	GCAEN	ADEXT	0	0	0	AD10	AD9	AD8
0x(FF)FF_8226	IIC2	IIC2SMB	FACK	ALERTE_N	SIICAEN	TCKSEL	SLTF	SHTF	0	0
0x(FF)FF_8227	IIC2	IIC2A2	SAD7	SAD6	SAD5	SAD4	SAD3	SAD2	SAD1	0
0x(FF)FF_8228	IIC2	IIC2SLTH	SSLT15	SSLT14	SSLT13	SSLT12	SSLT11	SSLT10	SSLT9	SSLT8
0x(FF)FF_8229	IIC2	IIC2SLTL	SSLT7	SSLT6	SSLT5	SSLT4	SSLT3	SSLT2	SSLT1	SSLT0
0x(FF)FF_822A	IIC2	IC2FLT	0	0	0	0	FLT3	FLT2	FLT1	FLT0
Address	Peripheral	Register	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_8240	MCG	MCGC1	CLKS			RDIV		IREFS	IRCLKEN	IREFSTE_N
0x(FF)FF_8241	MCG	MCGC2	BDIV	RANGE	HGO	LP	EREFs	ERCLKEN	EREFSTE_N	
0x(FF)FF_8242	MCG	MCGTRM				TRIM				
0x(FF)FF_8243	MCG	MCGSC	LOLS	LOCK	PLLST	IREFST	CLKST	OSCINIT	FTRIM	

## Memory

**Table 4-3. Detailed Peripheral Memory Map (continued)**

0x(FF)FF_8244	MCG	MCGC3	LOLIE	PLLS	CME	DIV32	VDIV			
0x(FF)FF_8245	MCG	MCGC4	0	0	DMX32	0	0	0	DRST_DRST	
0x(FF)FF_8246	MCG	RESERVED	—	—	—	—	—	—	—	—
Address	Peripheral	Register	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_8260	TPM1	TPM1SC	TOF	TOIE	CPWMS	CLKS		PS		
0x(FF)FF_8261	TPM1	TPM1CNTH	Bit 15	14	13	12	11	10	9	Bit 8
0x(FF)FF_8262	TPM1	TPM1CNTL	Bit 7	6	5	4	3	2	Bit 1	Bit 0
0x(FF)FF_8263	TPM1	TPM1MODH	Bit 15	14	13	12	11	10	9	Bit 8
0x(FF)FF_8264	TPM1	TPM1MODL	Bit 7	6	5	4	3	2	Bit 1	Bit 0
0x(FF)FF_8265	TPM1	TPM1C0SC	CH0F	CHOIE	MS0B	MS0A	ELS0B	ELS0A	0	0
0x(FF)FF_8266	TPM1	TPM1C0VH	Bit 15	14	13	12	11	10	9	Bit 8
0x(FF)FF_8267	TPM1	TPM1C0VL	Bit 7	6	5	4	3	2	Bit 1	Bit 0
0x(FF)FF_8268	TPM1	TPM1C1SC	CH1F	CH1IE	MS1B	MS1A	ELS1B	ELS1A	0	0
0x(FF)FF_8269	TPM1	TPM1C1VH	Bit 15	14	13	12	11	10	9	Bit 8
0x(FF)FF_826A	TPM1	TPM1C1VL	Bit 7	6	5	4	3	2	Bit 1	Bit 0
0x(FF)FF_826B	TPM1	TPM1C2SC	CH2F	CH2IE	MS2B	MS2A	ELS2B	ELS2A	0	0
0x(FF)FF_826C	TPM1	TPM1C2VH	Bit 15	14	13	12	11	10	9	Bit 8
0x(FF)FF_826D	TPM1	TPM1C2VL	Bit 7	6	5	4	3	2	Bit 1	Bit 0
Address	Peripheral	Register	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_8280	TPM2	TPM2SC	TOF	TOIE	CPWMS	CLKS		PS		
0x(FF)FF_8281	TPM2	TPM2CNTH	Bit 15	14	13	12	11	10	9	Bit 8
0x(FF)FF_8282	TPM2	TPM2CNTL	Bit 7	6	5	4	3	2	Bit 1	Bit 0
0x(FF)FF_8283	TPM2	TPM2MODH	Bit 15	14	13	12	11	10	9	Bit 8
0x(FF)FF_8284	TPM2	TPM2MODL	Bit 7	6	5	4	3	2	Bit 1	Bit 0
0x(FF)FF_8285	TPM2	TPM2C0SC	CH0F	CHOIE	MS0B	MS0A	ELS0B	ELS0A	0	0
0x(FF)FF_8286	TPM2	TPM2C0VH	Bit 15	14	13	12	11	10	9	Bit 8
0x(FF)FF_8287	TPM2	TPM2C0VL	Bit 7	6	5	4	3	2	Bit 1	Bit 0
0x(FF)FF_8288	TPM2	TPM2C1SC	CH1F	CH1IE	MS1B	MS1A	ELS1B	ELS1A	0	0
0x(FF)FF_8289	TPM2	TPM2C1VH	Bit 15	14	13	12	11	10	9	Bit 8
0x(FF)FF_828A	TPM2	TPM2C1VL	Bit 7	6	5	4	3	2	Bit 1	Bit 0
0x(FF)FF_828B	TPM2	TPM2C2SC	CH2F	CH2IE	MS2B	MS2A	ELS2B	ELS2A	0	0
0x(FF)FF_828C	TPM2	TPM2C2VH	Bit 15	14	13	12	11	10	9	Bit 8
0x(FF)FF_828D	TPM2	TPM2C2VL	Bit 7	6	5	4	3	2	Bit 1	Bit 0
Address	Peripheral	Register	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_82A0	MTIM1	MTIM1SC	TOF	TOIE	TRST	TSTP	0	0	0	0
0x(FF)FF_82A1	MTIM1	MTIM1CLK	0	0	CLKS		PS			
0x(FF)FF_82A2	MTIM1	MTIM1CNT	COUNT							

**Table 4-3. Detailed Peripheral Memory Map (continued)**

0x(FF)FF_82A3	MTIM1	MTIM1MOD	MOD															
Address	Peripheral	Register	Bit 7	6	5	4	3	2	1	Bit 0								
0x(FF)FF_82C0	RTC	RTCSC	RTIF	RTCLKS		RTIE	RTCPS											
0x(FF)FF_82C1	RTC	RTCCNT	RTCCNT															
0x(FF)FF_82C2	RTC	RTCMOD	RTCMOD															
Address	Peripheral	Register	Bit 7	6	5	4	3	2	1	Bit 0								
0x(FF)FF_82E0	FTSR	FCDIV	FDIVLD	PRDIV8	FDIV													
0x(FF)FF_82E1	FTSR	FOPT	KEYEN		0	0	0	0	SEC									
0x(FF)FF_82E2	FTSR	RESERVED	—	—	—	—	—	—	—	—								
0x(FF)FF_82E3	FTSR	FCNFG	CBEIE	CCIE	KEYACC	0	0	0	0	0								
0x(FF)FF_82E4	FTSR	FPROT	FPS						FPOOPEN									
0x(FF)FF_82E5	FTSR	FSTAT	FCBEF	FCCF	FPVIOL	FACCR	0	FBLANK	0	0								
0x(FF)FF_82E6	FTSR	FCMD	0	FCMD														
0x(FF)FF_82E7	FTSR	RESERVED	ERASE	PROG	IFREN	NVSTR	XE	YE	SE	MAS1								
0x(FF)FF_82E8	FTSR	RESERVED	—															
0x(FF)FF_82EF			—															
Address	Peripheral	Register	Bit 7	6	5	4	3	2	1	Bit 0								
0x(FF)FF_8300	MTIM2	MTIM2SC	TOF	TOIE	TRST	TSTP	0	0	0	0								
0x(FF)FF_8301	MTIM2	MTIM2CLK	0	0	CLKS		PS											
0x(FF)FF_8302	MTIM2	MTIM2CNT	COUNT															
0x(FF)FF_8303	MTIM2	MTIM2MOD	MOD															
Address	Peripheral	Register	Bit 31/23/15/7	30/22/14/6	29/21/13/5	28/20/12/4	27/19/11/3	26/18/10/2	25/17/9/1	Bit 24/16/8/0								
0x(FF)FF_E000	FEC	FEC_ID	FEC_ID[15:8]															
			FEC_ID[7:0]															
			0	0	0	0	0	DMA	FIFO	SMII								
			FEC_REV															
0x(FF)FF_E004	FEC	FEC_EIR	HBERR	BABR	BABT	GRA	TXF	TXB	RXF	RXB								
			MII	EBERR	LC	RL	UN	0	0	0								
			0	0	0	0	0	0	0	0								
			0	0	0	0	0	0	0	0								
0x(FF)FF_E008	FEC	FEC_EIMR	HBERR	BABR	BABT	GRA	TXF	TXB	RXF	RXB								
			MII	EBERR	LC	RL	UN	0	0	0								
			0	0	0	0	0	0	0	0								
			0	0	0	0	0	0	0	0								
0x(FF)FF_E010	FEC	FEC_RDAR	0	0	0	0	0	0	0	RDAR								
			0	0	0	0	0	0	0	0								

**Table 4-3. Detailed Peripheral Memory Map (continued)**

			0	0	0	0	0	0	0	0						
			0	0	0	0	0	0	0	0						
0x(FF)FF_E014	FEC	FEC_TDAR	0	0	0	0	0	0	0	TDAR						
			0	0	0	0	0	0	0	0						
			0	0	0	0	0	0	0	0						
0x(FF)FF_E018	FEC	FEC_R DES ACT IVE_CL	BYTE0													
			BYTE1													
			BYTE2													
			BYTE3													
0x(FF)FF_E01C	FEC	FEC_X DES ACT IVE_CL	BYTE0													
			BYTE1													
			BYTE2													
			BYTE3													
0x(FF)FF_E020	FEC	FEC_IEVENT_SE T	0	0	0	0	TFINT_E N	TXB_EN	RFINT_E N	RXB_EN						
			0	0	0	0	0	0	0	0						
			0	0	0	0	0	0	0	0						
			0	0	0	0	0	0	0	0						
0x(FF)FF_E024	FEC	FEC_ECR	TAG				0	TEST-MD	0	0						
			0	0	0	0	0	0	0	0						
			0	0	0	0	0	0	0	0						
			0	0	0	0	0	ETHER_E N	RESET							
0x(FF)FF_E040	FEC	FEC_MMFR	ST		OP		PA[4:1]									
			PA[0]		RA				TA							
			DATA[15:8]													
			DATA[7:0]													
0x(FF)FF_E044	FEC	FEC_MSCR	0	0	0	0	0	0	0	0						
			0	0	0	0	0	0	0	0						
			0	0	0	0	0	0	0	0						
			DIS_PRE	MII_SPEED						0						
0x(FF)FF_E048	FEC	FEC_MII_STATUS	0	0	0	0	FRAME_BIT_COUNT									
			0	0	CLK_CNTR											
			0	0	0	0	0	0	0	0						
			0	0	0	0	0	MII_DATAIO_STATE								
0x(FF)FF_E060 - 0x(FF)FF_E067	FEC	RESERVED	—													

**Table 4-3. Detailed Peripheral Memory Map (continued)**

0x(FF)FF_E080	FEC	FEC_R_ACTIVATE	R_ACTIVE	0	0	0	0	0	0	0	0
			0	0	0	R_STATE					
			0	0	0	0	0	0	0	0	0
			0	0	0	0	0	0	0	0	0
0x(FF)FF_E084	FEC	FEC_RCR	0	0	0	0	0	MAX_FL[10:8]			
				MAX_FL[7:0]							
			0	0	0	0	0	0	0	0	0
			0	0	FCE	BC_REJ	PROM	MII_MODE	DRT	LOOP	
0x(FF)FF_E088	FEC	FEC_R_HASH	FCE_DC	MUL_TC AST	HASH						
			0	0	0	0	0	0	0	0	0
			0	0	0	0	0	0	0	0	0
			0	0	0	0	0	0	0	0	0
0x(FF)FF_E08C	FEC	FEC_R_DATA			R_DATA[31:24]						
					R_DATA[23:16]						
					R_DATA[15:8]						
					R_DATA[7:0]						
0x(FF)FF_E090	FEC	FEC_AR_DONE	AR_HM_B	AR_EM_B	0	0	0	0	0	0	0
			0	0	0	0	0	0	0	0	0
			0	0	0	0	0	0	0	0	0
			0	0	0	0	0	0	0	0	0
0x(FF)FF_E094	FEC	FEC_R_TEST	0	0	0	0	R_DATA_AV	BABR	R_ADDR_ARM	0	
			0	0	0	0	0	0	0	0	R_TEST
			0	0	0	0	0	0	0	0	0
			0	0	0	0	0	0	0	0	0
0x(FF)FF_E098	FEC	RESERVED			—						
0x(FF)FF_E09B					—						
0x(FF)FF_E09C	FEC	FEC_R_DA_LOW			R_DA_BYTE_0						
					R_DA_BYTE_1						
					R_DA_BYTE_2						
					R_DA_BYTE_3						
0x(FF)FF_E0A0	FEC	FEC_R_DA_HIGH			R_DA_BYTE_4						
					R_DA_BYTE_5						
			0	0	0	0	0	0	0	0	0
			0	0	0	0	0	0	0	0	0
0x(FF)FF_E0C0	FEC	FEC_X_ACTIVATE	X_ACTIVE	0	0	0	0	0	0	0	0
			0	0	0	0	0	0	0	0	0

**Table 4-3. Detailed Peripheral Memory Map (continued)**

			0	0	0	0	0	0	0	0
			0	0	0	0	0	0	0	0
0x(FF)FF_E0C4	FEC	FEC_TCR	0	0	0	0	0	0	0	0
			0	0	0	0	0	0	0	0
			0	0	0	0	0	0	0	0
			0	0	0	RFC_PAUSE	TFC_PAUSE	FDEN	HBC	GTS
0x(FF)FF_E0C8	FEC	FEC_BACKOFF	0	0	0	0	0	0	0	0
			0	RANDOM[9:3]						
				RANDOM[2:0]			0	0	0	0
				0	0	0	0	0	0	0
0x(FF)FF_E0CC	FEC	FEC_X_DATA		X_DATA[31:24]						
				X_DATA[23:16]						
				X_DATA[15:8]						
				X_DATA[7:0]						
0x(FF)FF_E0D0	FEC	FEC_X_STATUS	0	0	0	0	0	0	DEF	HB
			LC	RL	RC				UN	CSL
			0	0	0	0	0	0	0	0
			0	0	0	0	0	0	0	0
0x(FF)FF_E0D4 - 0x(FF)FF_E0D7	FEC	RESERVED		—						
0x(FF)FF_E0D8	FEC	FEC_X_TEST	0	HBERR	BABT	GRA	X_SPACE_AV	X_DONE	X_ACCE	X_EARLY_CLSN
			0	0	0	0	0	0	0	X_TEST
			0	0	0	0	0	0	0	0
			0	0	0	0	0	0	COLL	SLOT
0x(FF)FF_E0DC	FEC	FEC_F_Dxfc_da <sub>1</sub>	0	0	0	0	0	0	0	1
			1	0	0	0	0	0	0	0
			1	1	0	0	0	0	1	0
			0	0	0	0	0	0	0	0
0x(FF)FF_E0E0	FEC	FEC_F_Dxfc_da <sub>2</sub>	0	0	0	0	0	0	0	0
			0	0	0	0	0	0	0	1
			0	0	0	0	0	0	0	0
0x(FF)FF_E0E4	FEC	FEC_PALR		PADDR1[31:24]						
				PADDR1[23:16]						
				PADDR1[15:8]						
				PADDR1[7:0]						

**Table 4-3. Detailed Peripheral Memory Map (continued)**

0x(FF)FF_E0E8	FEC	FEC_PAUR	PADDR2[15:8]							
			PADDR2[7:0]							
			TYPE[15:8]							
			TYPE[7:0]							
0x(FF)FF_E0EC	FEC	FEC_OPD	OPCODE[15:8]							
			OPCODE[7:0]							
			PAUSE_DUR[15:8]							
			PAUSE_DUR[7:0]							
0x(FF)FF_E100	FEC	FEC_D_INSTR_REG	INSTR[14:7]							
			INSTR[6:0]							
			0	0	0	0	0	0	0	0
			PC							
0x(FF)FF_E104	FEC	FEC_D_CONTEXT_REG	0	0	0	0	0	TCONTEXT		
			0	0	0	0	0	RCONTEXT		
			0	0	0	0	0	0	0	0
			0	0	0	0	0	0	0	Z_FLAG
0x(FF)FF_E108	FEC	FEC_D_TEST_CTRL	TEST_MODE	READ_ROM	HOLD	0	0	0	0	0
			0	0	0	0	0	0	0	0
			0	0	0	0	0	0	0	0
			0	0	0	0	0	0	0	0
0x(FF)FF_E10C	FEC	FEC_D_ACC_REG	ACC_REG[31:24]							
			ACC_REG[23:16]							
			ACC_REG[15:8]							
			ACC_REG[7:0]							
0x(FF)FF_E110	FEC	FEC_D_ONES	1	1	1	1	1	1	1	1
			1	1	1	1	1	1	1	1
			1	1	1	1	1	1	1	1
			1	1	1	1	1	1	1	1
0x(FF)FF_E114	FEC	FEC_D_ZEROS	0	0	0	0	0	0	0	0
			0	0	0	0	0	0	0	0
			0	0	0	0	0	0	0	0
			0	0	0	0	0	0	0	0
0x(FF)FF_E118	FEC	FEC_IAUR	IAUR[31:24]							
			IAUR[23:16]							
			IAUR[15:8]							
			IAUR[7:0]							

## Memory

**Table 4-3. Detailed Peripheral Memory Map (continued)**

0x(FF)FF_E11C	FEC	FEC_IALR	IALR[31:24]							
			IALR[24:16]							
			IALR[15:8]							
			IALR[7:0]							
0x(FF)FF_E120	FEC	FEC_GAUR	GAUR[31:24]							
			GAUR[23:16]							
			GAUR[15:8]							
			GAUR[7:0]							
0x(FF)FF_E124	FEC	FEC_GALR	GALR[31:24]							
			GALR[23:17]							
			GALR[15:8]							
			GALR[7:0]							
0x(FF)FF_E128	FEC	FEC_RANDOM	RANDOM[31:24]							
			RANDOM[23:16]							
			RANDOM[15:8]							
			RANDOM[7:0]							
0x(FF)FF_E12C	FEC	FEC_RAND1	RAND1[31:24]							
			RAND1[23:16]							
			RAND1[15:8]							
			RAND1[7:0]							
0x(FF)FF_E130	FEC	FEC_TMP	TMP[31:24]							
			TMP[23:16]							
			TMP[15:8]							
			TMP[7:0]							
0x(FF)FF_E140	FEC	FEC_FIFO_ID	FIFO_REV							
			0	0	0	0	0	0	0	0
			FIFO_SZ[15:8]							
			FIFO_SZ[7:2]							
0x(FF)FF_E144	FEC	FEC_TFWR	0	0	0	0	0	0	0	0
			0	0	0	0	0	0	0	0
			0	0	0	0	0	0	0	0
			0	0	0	0	0	0	TFWR	
0x(FF)FF_E148	FEC	FEC_FCNTRL	FC_TEST	0	X_SPACE_AVAIL	X_DATA_AVAIL	R_SPACE_AVAIL	R_DATA_AVAIL	0	0
			0	0	0	0	0	0	0	0
			0	0	0	0	0	0	XBYTE_COUNT[9:8]	
			XBYTE_COUNT[7:2]							
0x(FF)FF_E14C	FEC	FEC_FRBR	0	0	0	0	0	0	0	0
			0	0	0	0	0	0	0	0

**Table 4-3. Detailed Peripheral Memory Map (continued)**

			0	0	0	0	0	1	R_BOUND[9:8]	
			R_BOUND[7:2]						0	0
0x(FF)FF_E150	FEC	FEC_FRSR	0	0	0	0	0	0	0	0
			0	0	0	0	0	0	0	0
			0	0	0	0	0	1	R_FSTART[9:8]	
			R_FSTART[7:2]						0	0
0x(FF)FF_E154	FEC	FEC_R_COUNT	0	0	0	R_COUNT				
			0	0	0	0	0	0	0	0
			0	0	0	0	0	0	0	0
			0	0	0	0	0	0	0	0
0x(FF)FF_E158	FEC	FEC_R_LAG	0	0	0	0	0	0	0	0
			0	0	0	0	0	0	0	0
			0	0	0	0	0	1	R_LAG[9:8]	
			R_LAG[7:2]						0	0
0x(FF)FF_E15C	FEC	FEC_R_READ	0	0	0	0	0	0	0	0
			0	0	0	0	0	0	0	0
			0	0	0	0	0	1	R_READ[9:8]	
			R_READ[7:2]						0	0
0x(FF)FF_E160	FEC	FEC_R_WRITE	0	0	0	0	0	0	0	0
			0	0	0	0	0	0	0	0
			0	0	0	0	0	1	R_WRITE[9:8]	
			R_WRITE[7:2]						0	0
0x(FF)FF_E164	FEC	FEC_X_COUNT	0	0	0	X_COUNT				
			0	0	0	0	0	0	0	0
			0	0	0	0	0	1	0	0
			0	0	0	0	0	0	0	0
0x(FF)FF_E168	FEC	FEC_X_LAG	0	0	0	0	0	0	0	0
			0	0	0	0	0	0	0	0
			0	0	0	0	0	1	X_LAG[9:8]	
			X_LAG[7:2]						0	0
0x(FF)FF_E16C	FEC	FEC_X_RETRY	X_RETRY[31:24]							
			X_RETRY[24:16]							
			X_RETRY[15:8]							
			X_RETRY[7:0]							
0x(FF)FF_E170	FEC	FEC_X_WRITE	0	0	0	0	0	0	0	0
			0	0	0	0	0	0	0	0
			0	0	0	0	0	1	X_WRITE[9:8]	
			X_WRITE[7:2]						0	0

## Memory

**Table 4-3. Detailed Peripheral Memory Map (continued)**

0x(FF)FF_E174	FEC	FEC_X_READ	0	0	0	0	0	0	0	0
			0	0	0	0	0	0	0	0
			0	0	0	0	0	1	X_READ[9:8]	
						X_READ[7:2]			0	0
0x(FF)FF_E180	FEC	FEC_E_RDSR				R_DES_START[31:24]				
						R_DES_START[23:16]				
						R_DES_START[15:8]				
						R_DES_START[15:2]		0	0	0
0x(FF)FF_E184	FEC	FEC_E_TDSR				X_DES_START[31:24]				
						X_DES_START[23:16]				
						X_DES_START[15:8]				
						X_DES_START[15:2]		0	0	0
0x(FF)FF_E188	FEC	FEC_EMRBR	0	0	0	0	0	0	0	0
			0	0	0	0	0	0	0	0
			0	0	0	0	0		R_BUF_SIZE[6:4]	
					R_BUF_SIZE[3:0]		0	0	0	0
0x(FF)FF_E18C - 0x(FF)FF_E7FF	FEC	RESERVED				—				
						—				
Address	Peripheral	Register	Bit 31/23/15/7	30/22/14/ 6	29/21/13/ 5	28/20/12/ 4	27/19/11/ 3	26/18/10/ 2	25/17/9/1	Bit 24/16/8/0
0x(FF)FF_E800	MB	MBCSAR0				BAM[31:24]				
						BAM[23:16]				
			0	0	0	0	0	0	0	0
			0	0	0	0	0	0	0	0
0x(FF)FF_E804	MB	MBCSMR0				BAM[31:24]				
						BAM[23:16]				
			0	0	0	0	0	0	0	WP
			0	0	0	0	0	0	0	V
0x(FF)FF_E808	MB	MBCSCR0	0	0	0	0	0	0	0	0
			0	0		ASET		RDAH		WRAH
						WS			MUX	AA
				PS	0	0	0	0	0	0
0x(FF)FF_E80C	MB	MBCSAR1				BAM[31:24]				
						BAM[23:16]				
			0	0	0	0	0	0	0	0
			0	0	0	0	0	0	0	0
0x(FF)FF_E810	MB	MBCSMR1				BAM[31:24]				
						BAM[23:16]				

**Table 4-3. Detailed Peripheral Memory Map (continued)**

			0	0	0	0	0	0	0	WP									
			0	0	0	0	0	0	0	V									
0x(FF)FF_E814	MB	MBCSCR1	0	0	0	0	0	0	0	0									
			0	0	ASET		RDAH		WRAH										
					WS				MUX	AA									
				PS		0	0	0	0	0									
Address	Peripheral	Register	Bit 15/7	14/6	13/5	12/4	11/3	10/2	9/1	Bit 8/0									
0x(FF)FF_FFCC	INTC	INTC_ORMR	0	0	0	0	0	0	0	FECDO									
0x(FF)FF_FFCD			0	0	SCI3DO	0	0	0	0	0									
Address	Peripheral	Register	Bit 7	6	5	4	3	2	1	Bit 0									
0x(FF)FF_FFD3	INTC	INTC_FRC	0	LVL1	LVL2	LVL3	LVL4	LVL5	LVL6	LVL7									
0x(FF)FF_FFD8	INTC	INTC_PL6P7	0	0	REQN														
0x(FF)FF_FFD9	INTC	INTC_PL6P6	0	0	REQN														
0x(FF)FF_FFDB	INTC	INTC_WCR	ENB	0	0	0	0	MASK											
0x(FF)FF_FFD8	INTC	INTC_SFRC	0	0	SET														
0x(FF)FF_FFD9	INTC	INTC_CFRC	0	0	CLR														
0x(FF)FF_FFE0	INTC	INTC_SWIACK	0	VECN															
0x(FF)FF_FFE4	INTC	INTC_LVL1IACK	0	VECN															
0x(FF)FF_FFE8	INTC	INTC_LVL2IACK	0	VECN															
0x(FF)FF_FFEC	INTC	INTC_LVL3IACK	0	VECN															
0x(FF)FF_FFF0	INTC	INTC_LVL4IACK	0	VECN															
0x(FF)FF_FFF4	INTC	INTC_LVL15ACK	0	VECN															
0x(FF)FF_FFF8	INTC	INTC_LVL6IACK	0	VECN															
0x(FF)FF_FFFC	INTC	INTC_LVL7IACK	0	VECN															

Recall that ColdFire has a big endian byte addressable memory architecture. The most significant byte of each address is the lowest numbered as shown in [Figure 4-3](#). Multi-byte operands (e.g., 16-bit words and 32-bit longwords) are referenced using an address pointing to the most significant (first) byte.

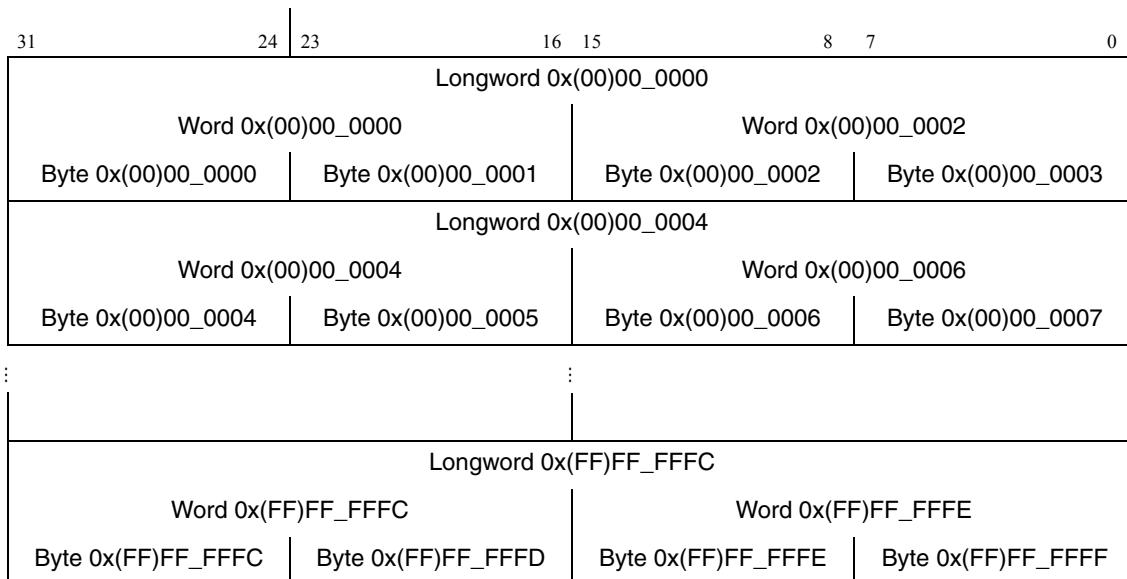


Figure 4-3. ColdFire Memory Organization

#### 4.2.1 Flash Module Reserved Memory Locations

Several reserved flash memory locations, shown in [Table 4-4](#), are used for storing values used by corresponding peripheral registers. These registers include an 8-byte backdoor key that can be used to gain access to secure memory resources. During reset events, the contents of the flash protection byte (NVPROT) and flash nonvolatile byte (NVOPT) in the reserved flash memory are transferred into the corresponding FPROT and FOPT registers in the high-page register area to control security and block protection options.

Table 4-4. Reserved Flash Memory Addresses

Address	MSB <sup>1</sup> (0x0)	(0x1)	(0x2)	LSB <sup>2</sup> (0x3)
0x(00)00_03FC	Reserved		FTRIM (bit 0)	TRIM
0x(00)00_0400	Backdoor comparison key bytes 0–3			
	byte0	byte1	byte2	byte3
0x(00)00_0404	Backdoor comparison key bytes 4–7			
	byte4	byte5	byte6	byte7
0x(00)00_0408	Reserved			
0x(00)00_040C	Reserved	NVPROT	Reserved	NVOPT

<sup>1</sup> MSB = most significant byte<sup>2</sup> LSB = least significant byte

**Table 4-5. Reserved Flash Memory Addresses**

Address	Register	7	6	5	4	3	2	1	0
0x(00)00_03FC– 0x(00)00_03FD	Reserved	—	—	—	—	—	—	—	—
0x(00)00_03FE	Storage of FTRIM	0	0	0	0	0	0	0	FTRIM
0x(00)00_03FF	Storage of MCGTRM	TRIM							
0x(00)00_0400– 0x(00)00_0407		8-Byte Backdoor Comparison Key							
0x(00)00_0408– 0x(00)00_040C	Reserved	—	—	—	—	—	—	—	—
0x(00)00_040D	NVPROT	FPS							FPOOPEN
0x(00)00_040E	Reserved	—	—	—	—	—	—	—	—
0x(00)00_040F	NVOPT	KEYEN		0	0	0	0	SEC	

The factory trim values are stored in the flash information row (IFR)<sup>1</sup> and are automatically loaded into the MCGTRM and MCGSC registers after any reset. The oscillator trim values stored in TRIM and FTRIM can be reprogrammed by third party programmers and must be copied into the corresponding MCG registers (MCGTRM and MCGSC) by user code to override the factory trim.

#### NOTE

When the MCU is in active BDM, the trim value in the IFR is not loaded. Instead, the MCGTRM register resets to 0x80 and MCGSC[FTRIM] resets to zero.

Provided the key enable (KEYEN) bit is set, the 8-byte comparison key can be used to temporarily disengage memory security. This key mechanism can be accessed only through user code running in secure memory (A security key cannot be entered directly through background debug commands). This security key can be disabled completely by clearing the KEYEN bit. If the security key is disabled, the only way to disengage security is by mass-erasing the flash (normally through the background debug interface) and verifying the flash is blank.

## 4.3 RAM

An MCF51CN128 series microcontroller includes up to 24 KB of static RAM. RAM is most efficiently accessed using the A5-relative addressing mode (address register indirect with displacement mode). Any single bit in this area can be accessed with the bit manipulation instructions (BCLR, BSET,etc.).

At power-on, the contents of RAM are uninitialized. RAM data is unaffected by any reset provided that the supply voltage does not drop below the minimum value for RAM retention ( $V_{RAM}$ ).

1. **IFR** — Nonvolatile information memory that can only be accessed during production test. During production test, system initialization, configuration, and test information is stored in the IFR. This information cannot be read or modified in normal user or background debug modes.

## 4.4 Flash Memory

The flash memory is intended primarily for program storage and read-only data. In-circuit programming allows the operating program to be loaded into the flash memory after final assembly of the application product. It is possible to program the entire array through the single-wire background debug interface. Because no special voltages are needed for flash erase and programming operations, in-application programming is also possible through other software-controlled communication paths.

Flash memory is ideal for single-supply applications allowing for field reprogramming without requiring external high voltage sources for program or erase operations. The flash module includes a memory controller that executes commands to modify flash memory contents.

Array read access time is one bus cycle for bytes, aligned words, and aligned longwords. Multiple accesses are needed for misaligned words and longword operands. For flash memory, an erased bit reads 1 and a programmed bit reads 0. It is not possible to read from a flash block while any command is executing on that specific flash block.

### CAUTION

A flash block address must be in the erased state before being programmed. Cumulative programming of bits within a flash block address is not allowed except for status field updates required in EEPROM emulation applications.

Flash memory on this device must be programmed 32-bits at a time when the low-voltage detect flag (LVDF) in the system power management status and control 1 register (SPMSC1) is clear. If SPMSC1[LVDF] is set, the programming sequence must be modified such that odd and even bytes are written separately. This device's flash memory is organized as two 16-bit wide blocks interleaved to yield a 32-bit data path. When programming flash when LVDF is set, alternate bytes must be set to 0xFF as shown in [Table 4-6](#). Failure to adhere to these guidelines may result in a partially programmed flash array.

**Table 4-6. Low-Voltage Programming Sequence Example**

Addresses	Desired Value	Values Programmed
0x00 – 0x03 0x00 – 0x03	0x5555_AAAA	0x55FF_AAFF 0xFF55_FFAA
0x04 – 0x07 0x04 – 0x07	0xCCCC_CCCC	0xCCFF_CCFF 0xFFCC_FFCC
0x08 – 0x0B 0x08 – 0x0B	0x1234_5678	0x12FF_56FF 0xFF34_FF78
0x0C – 0x0F 0x0C – 0x0F	0x9ABC_DEF0	0x9AFF_DEF0 0xFFBC_FFF0

### 4.4.1 Features

Features of the flash memory include:

- Flash size
  - MCF51CN128: 131,072 bytes (128 sectors of 1024 bytes each)
- Automated program and erase algorithm

- Fast program and sector erase operation
- Burst program command for faster flash array program times
- Single power supply program and erase
- Command interface for fast program and erase operation
- Up to 100,000 program/erase cycles at typical voltage and temperature
- Flexible block protection (on any 2 KB memory boundary)
- Security feature to prevent unauthorized access to on-chip memory and resources
- Auto power-down for low-frequency read accesses

## 4.4.2 Register Descriptions

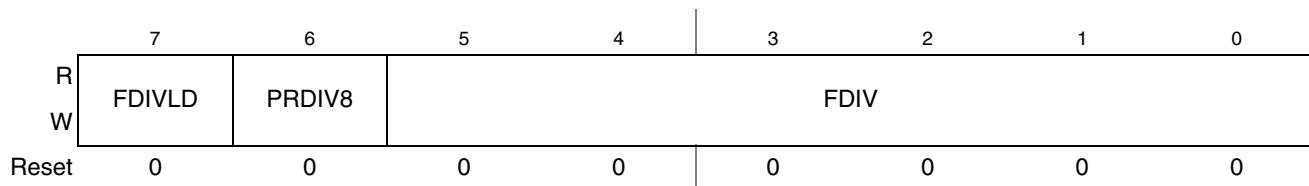
The flash controller (FTSR) contains a set of 16 control and status registers as shown in Table 4-7. Detailed descriptions of each register bit are provided in the following sections.

**Table 4-7. FTSR Register Memory Map and Bit Locations**

Address	Register	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_82E0	FCDIV	FDIVLD	PRDIV8	FDIV					
0x(FF)FF_82E1	FOPT	KEYEN			0	0	0	0	SEC
0x(FF)FF_82E2	RESERVED FRSV0	—	—	—	—	—	—	—	—
0x(FF)FF_82E3	FCNFG	CBEIE	CCIE	KEYACC	0	0	0	0	0
0x(FF)FF_82E4	FPPROT	FPS						FPOPE N	
0x(FF)FF_82E5	FSTAT	FCBEF	FCCF	FPVIOL	FACCERR	0	FBLANK	0	0
0x(FF)FF_82E6	FCMD	0	FCMD						
0x(FF)FF_82E7	RESERVED	—	—	—	—	—	—	—	—
0x(FF)FF_82EF									

### 4.4.2.1 Flash Clock Divider Register (FCDIV)

The FCDIV register controls the length of timed events in program and erase algorithms executed by the flash memory controller. All bits in the FCDIV register are readable and writable with restrictions as determined by the value of FDIVLD when writing to the FCDIV register.



**Figure 4-4. Flash Clock Divider Register (FCDIV)**

**Table 4-8. FCDIV Field Descriptions**

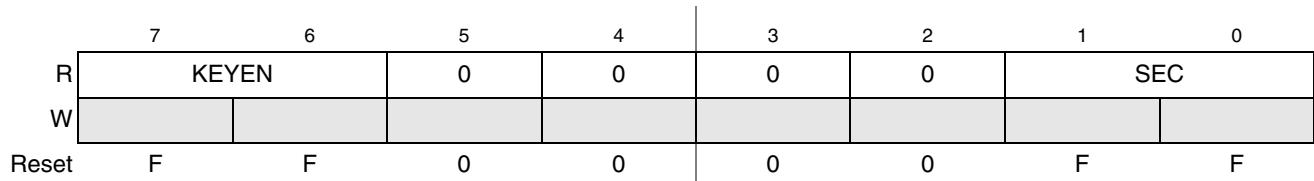
Field	Description
7 FDIVLD	<b>Clock Divider Load Control.</b> When writing to the FCDIV register for the first time after a reset, the value written to this bit controls the future ability to write to the FCDIV register: 0 Locks the FCDIV register contents; all future writes to FCDIV are ignored. 1 Keeps the FCDIV register writable; next write to FCDIV is allowed. When reading the FCDIV register, the value of the FDIVLD bit read indicates the following: 0 FCDIV register has not been written to since the last reset. 1 FCDIV register has been written to since the last reset.
6 PRDIV8	<b>Enable Prescalar by 8.</b> 0 The bus clock is directly fed into the clock divider. 1 The bus clock is divided by 8 before feeding into the clock divider.
5–0 FDIV	<b>Clock Divider Bits.</b> The combination of PRDIV8 and FDIV[5:0] must divide the bus clock to a frequency of 150 kHz–200 kHz. The minimum divide ratio is 2 (PRDIV8=0, FDIV=0x01) and the maximum divide ratio is 512 (PRDIV8=1, FDIV=0x3F).

#### 4.4.2.2 Flash Options Register (FOPT and NVOPT)

The FOPT register holds all bits associated with the security of the MCU and flash module. The FOPT register is read only.

The FOPT register is loaded from the flash configuration field during the reset sequence, indicated by F in Figure 4-5.

The security feature in the flash module is described in [Section 4.5, “Security.”](#)

**Figure 4-5. Flash Options Register (FOPT)****Table 4-9. FOPT Field Descriptions**

Field	Description
7–6 KEYEN	<b>Backdoor Key Security Enable Bits.</b> The KEYEN[1:0] bits define the enabling of backdoor key access to the flash module. 00 Disabled 01 Disabled (Preferred KEYEN state to disable Backdoor Key Access) 10 Enabled 11 Disabled
5–2	Reserved, must be cleared.
1–0 SEC	<b>Flash Security Bits.</b> The SEC[1:0] bits define the security state of the MCU. If the flash module is unsecured using backdoor key access, the SEC[1:0] bits are forced to the unsecured state. 00 Unsecured 01 Unsecured 10 Secured 11 Unsecured

#### 4.4.2.3 Flash Configuration Register (FCNFG)

The FCNFG register gates the security backdoor writes.

KEYACC is readable and writable while all remaining bits read 0 and are not writable. KEYACC is only writable if KEYEN is set to the enabled state (see [Section 4.4.2.2, “Flash Options Register \(FOPT and NVOPT\)”](#)).

##### NOTE

Flash array reads are allowed while KEYACC is set.

	7	6	5	4	3	2	1	0
R	0	0	KEYACC	0	0	0	0	0
W								
Reset	0	0	0	0	0	0	0	0

**Figure 4-6. Flash Configuration Register (FCNFG)**

**Table 4-10. FCNFG Field Descriptions**

Field	Description
7–6	Reserved, must be cleared.
5 KEYACC	Enable Security Key Writing 0 Writes to the flash block are interpreted as the start of a command write sequence. 1 Writes to the flash block are interpreted as keys to open the backdoor.
4–0	Reserved, must be cleared.

#### 4.4.2.4 Flash Protection Register (FPROT and NVPROT)

The FPROT register defines which flash sectors are protected against program or erase operations. FPROT bits are readable and writable if the size of the protected flash memory is being increased. Any write to FPROT that attempts to decrease the size of the protected flash memory is ignored.

During the reset sequence, the FPROT register is loaded from the flash protection byte in the flash configuration field, indicated by F in [Figure 4-7](#). To change the flash protection loaded during the reset sequence, the flash sector containing the flash configuration field must be unprotected. Then, the flash protection byte must be reprogrammed.

Trying to alter data in any protected area in the flash memory results in a protection violation error and FSTAT[FPVIOL] is set. The mass erase of the flash array is not possible if any of the flash sectors contained in the flash array are protected.

	7	6	5	4	3	2	1	0
R	F	F	F	F	F	F	F	F
W				FPS				FPOOPEN
Reset	F	F	F	F	F	F	F	F

**Figure 4-7. Flash Protection Register (FPROT)**

**Table 4-11. FPROT Field Descriptions**

Field	Description
7–1 FPS	<b>Flash Protection Size.</b> With FPOpen set, the FPS bits determine the size of the protected flash address range as shown in <a href="#">Table 4-12</a> .
0 FPOpen	Flash Protection Open 0 Flash array fully protected. 1 Flash array protected address range determined by FPS bits.

**Table 4-12. Flash Protection Address Range**

FPS	FPOpen	Protected Address Range Relative to Flash Array Base	Protected Size
—	1	0x0_0000–0x1_FFFF	128 KB
0x00–0x3F		0x0_0000–0x1_FFFF	128 KB
0x40		0x0_0000–0x1_F7FF	126 KB
0x41		0x0_0000–0x1_EFFF	124 KB
0x42		0x0_0000–0x1_E7FF	122 KB
0x43		0x0_0000–0x1_DFFF	120 KB
0x44		0x0_0000–0x1_D7FF	118 KB
0x45		0x0_0000–0x1_CFFF	116 KB
0x46		0x0_0000–0x1_C7FF	114 KB

**Table 4-12. Flash Protection Address Range (continued)**

FPS	FPOOPEN	Protected Address Range Relative to Flash Array Base	Protected Size
0x47	1	0x0_0000–0x1_BFFF	112 KB
...		...	...
0x5B		0x0_0000–0x1_1FFF	72 KB
0x5C		0x0_0000–0x1_17FF	70 KB
0x5D		0x0_0000–0x1_0FFF	68 KB
0x5E		0x0_0000–0x1_07FF	66 KB
0x5F		0x0_0000–0x0_FFFF	64 KB
0x60		0x0_0000–0x0_F7FF	62 KB
0x61		0x0_0000–0x0_EFFF	60 KB
0x62		0x0_0000–0x0_E7FF	58 KB
0x63		0x0_0000–0x0_DFFF	56 KB
...		...	...
0x77		0x0_0000–0x0_3FFF	16 KB
0x78		0x0_0000–0x0_37FF	14 KB
0x79		0x0_0000–0x0_2FFF	12 KB
0x7A		0x0_0000–0x0_27FF	10 KB
0x7B		0x0_0000–0x0_1FFF	8 KB
0x7C		0x0_0000–0x0_17FF	6 KB
0x7D		0x0_0000–0x0_0FFF	4 KB
0x7E		0x0_0000–0x0_07FF	2 KB
0x7F		No Protection	0 KB

#### 4.4.2.5 Flash Status Register (FSTAT)

The FSTAT register defines the operational status of the flash module. FCBEF, FPVIOL and FACCERR are readable and writable. FBLANK is read only. The remaining bits read 0 and are not writable.

	7	6	5	4		3	2	1	0
R	FCBEF	FCCF	FPVIOL	FACCERR	0	FBLANK	0	0	0
W	w1c		w1c	w1c	0	0	0	0	0
Reset	1	1	0	0	0	0	0	0	0

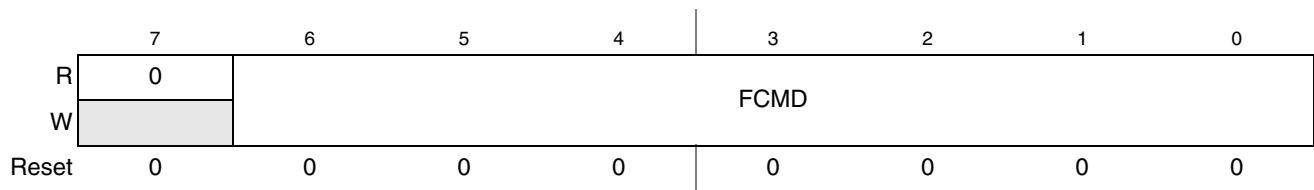
**Figure 4-8. Flash Status Register (FSTAT)**

**Table 4-13. FSTAT Field Descriptions**

Field	Description
7 FCBEF	<b>Command Buffer Empty Flag.</b> The FCBEF flag indicates that the command buffer is empty so that a new command write sequence can be started when performing burst programming. Writing a 0 to the FCBEF flag has no effect on FCBEF. Writing a 0 to FCBEF after writing an aligned address to the flash array memory, but before FCBEF is cleared, aborts a command write sequence and causes the FACCERR flag to set. Writing a 0 to FCBEF outside of a command write sequence does not set the FACCERR flag. Writing a 1 to this bit clears it. 0 Command buffers are full. 1 Command buffers are ready to accept a new command.
6 FCCF	<b>Command Complete Flag.</b> The FCCF flag indicates that there are no more commands pending. The FCCF flag is cleared when FCBEF is cleared and sets automatically after completion of all active and pending commands. The FCCF flag does not set when an active program command completes and a pending burst program command is fetched from the command buffer. Writing to the FCCF flag has no effect on FCCF. 0 Command in progress. 1 All commands are completed.
5 FPVIOL	<b>Protection Violation Flag.</b> The FPVIOL flag indicates an attempt was made to program or erase an address in a protected area of the flash memory during a command write sequence. Writing a 0 to the FPVIOL flag has no effect on FPVIOL. Writing a 1 to this bit clears it. While FPVIOL is set, it is not possible to launch a command or start a command write sequence. 0 No protection violation detected. 1 Protection violation has occurred.
4 FACCERR	<b>Access Error Flag.</b> The FACCERR flag indicates an illegal access has occurred to the flash memory caused by either a violation of the command write sequence, issuing an illegal flash command (see <a href="#">Section 4.4.2.6, “Flash Command Register (FCMD)</a> ”), or the execution of a CPU STOP instruction while a command is executing (FCCF = 0). Writing a 0 to the FACCERR flag has no effect on FACCERR. Writing a 1 to this bit clears it. While FACCERR is set, it is not possible to launch a command or start a command write sequence. 0 No access error detected. 1 Access error has occurred.
3	Reserved, must be cleared.
2 FBLANK	<b>Flag Indicating the Erase Verify Operation Status.</b> When the FCCF flag is set after completion of an erase verify command, the FBLANK flag indicates the result of the erase verify operation. The FBLANK flag is cleared by the flash module when FCBEF is cleared as part of a new valid command write sequence. Writing to the FBLANK flag has no effect on FBLANK. 0 Flash block verified as not erased. 1 Flash block verified as erased.
1–0	Reserved, must be cleared.

#### 4.4.2.6 Flash Command Register (FCMD)

The FCMD register is the flash command register. All FCMD bits are readable and writable during a command write sequence while bit 7 reads 0 and is not writable.

**Figure 4-9. Flash Command Register (FCMD)**

**Table 4-14. FCMD Field Descriptions**

Field	Description
7	Reserved, must be cleared.
6–0 FCMD	<b>Flash Command.</b> Valid flash commands are shown below. Writing any command other than those listed sets the FACCERR flag in the FSTAT register. 0x05 Erase Verify 0x20 Program 0x25 Burst Program 0x40 Sector Erase 0x41 Mass Erase

## 4.5 Security

The MCF51CN128 series microcontroller includes circuitry to prevent unauthorized access to the contents of flash and RAM memory. When security is engaged, BDM access is restricted to the upper byte of the ColdFire CSR, XCSR, and CSR2 registers. RAM, flash memory, peripheral registers and most of the CPU register set are not available via BDM. Programs executing from internal memory have normal access to all microcontroller memory locations and resources.

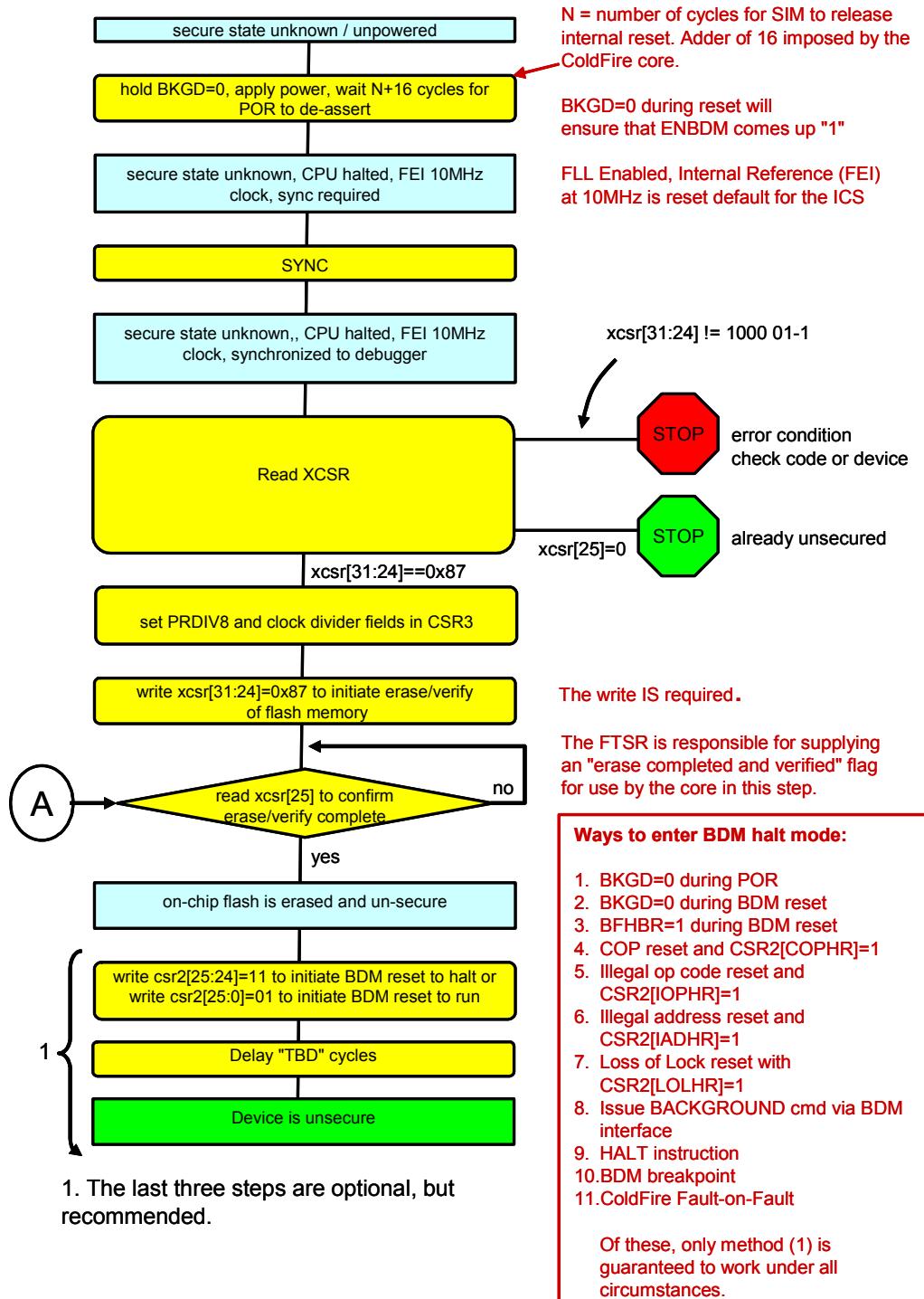
Security is engaged or disengaged based on the state of two nonvolatile register bits (SEC01, SEC00) in the FOPT register. During reset, the contents of the nonvolatile location, NVOPT, are copied from flash into the working FOPT register in register space. Enable security by programming the NVOPT location which can be done at the same time the flash memory is programmed. Set NVOPT[SEC01, SEC00] to 10 engage security. The other three combinations disengage security.

After exiting reset, the XCSR[SEC] bit in the ColdFire debug module is set if the device is secured, cleared otherwise.

You can choose to allow or disallow a security unlocking mechanism through an 8-byte backdoor security key. The security key can be written by the CPU executing from internal memory. It cannot be entered without the cooperation of a secure user program.

Development tools unsecure devices by an alternate BDM-based methodology shown in [Figure 4-10](#). Because both RESET and BKGD pins can be reprogrammed by software, a power-on-reset is required to be certain of obtaining control of the device via BDM, which is a required prerequisite for clearing security. Other methods (outlined in red in [Figure 4-10](#)) can also be used, but may not work under all circumstances.

This device supports two levels of security. Both restrict BDM communications as outlined above. In addition, SOPT1[SL] (see [Section 5.7.3, “System Options 1 Register \(SOPT1\)”](#)) can be used to enable/disable off-chip data accesses through the Mini-FlexBus interface. Off-chip op code accesses through the Mini-FlexBus are always disallowed when security is enabled.



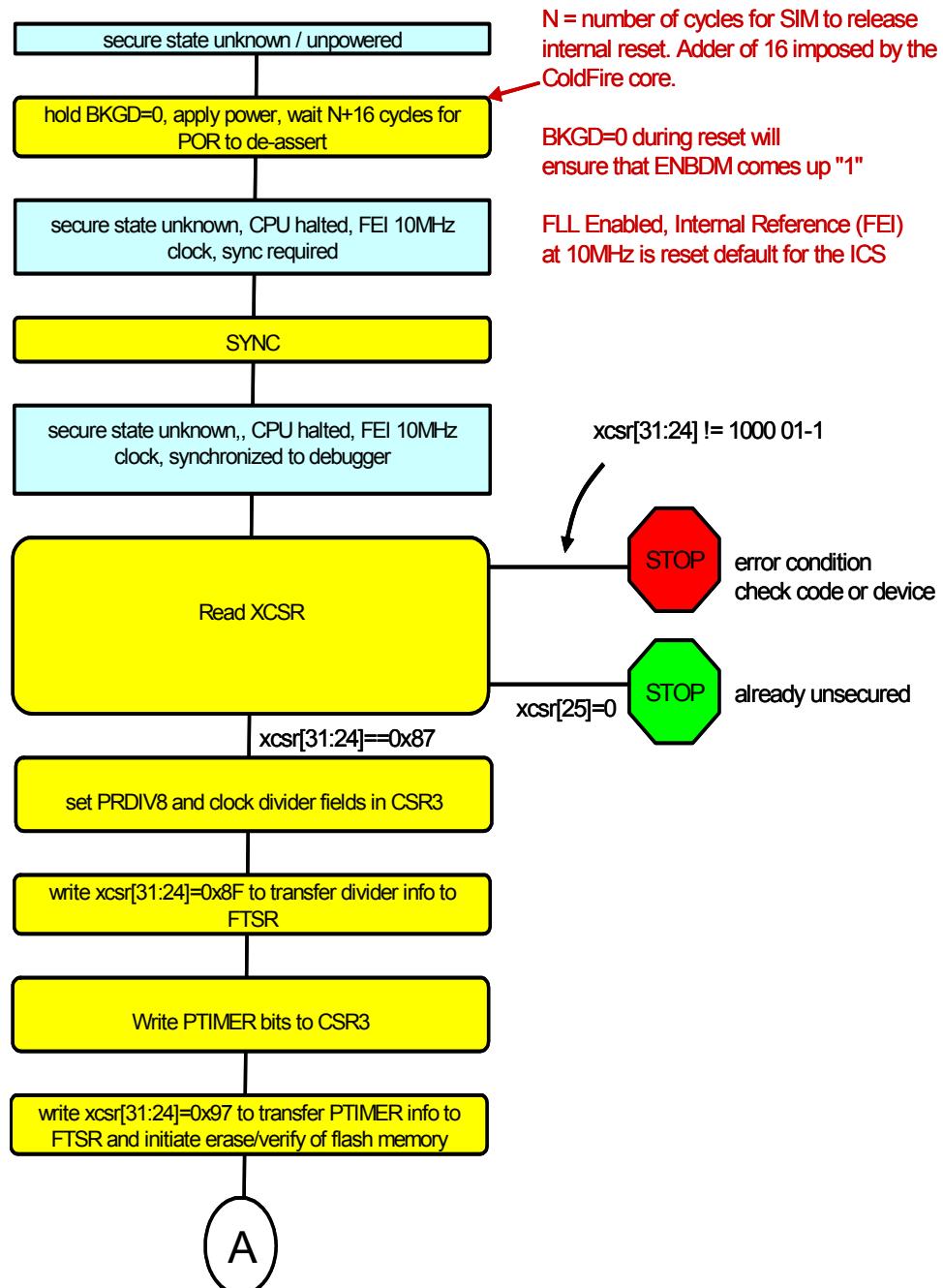


Figure 4-10. Procedure for Clearing Security on MCF51CN128 Series MCUs via the BDM Port

# Chapter 5

## Resets, Interrupts, and General System Control

### 5.1 Introduction

This section discusses basic reset and interrupt mechanisms and the various sources of reset and interrupt on an MCF51CN128 series microcontroller. Some interrupt sources from peripheral modules are discussed in greater detail within other sections of this document. This section gathers basic information about all reset and interrupt sources in one place for easy reference. A few reset and interrupt sources, including the computer operating properly (COP) watchdog are not part of on-chip peripheral systems with their own chapters.

### 5.2 Features

Reset and interrupt features include:

- Multiple sources of reset for flexible system configuration and reliable operation
- System reset status (SRS) register to indicate source of most recent reset
- Separate interrupt vector for most modules (reduces polling overhead) (see [Table 5-1](#))

### 5.3 Microcontroller Reset

Resetting the microcontroller provides a way to start processing from a known set of initial conditions. When the ColdFire processor exits reset, it fetches initial 32-bit values for the supervisor stack pointer and program counter from locations 0x(00)00\_0000 and 0x(00)00\_0004, respectively. On-chip peripheral modules are disabled and I/O pins are initially configured as general-purpose high-impedance inputs with pull-up devices disabled.

The MCF51CN128 series microcontrollers have the following sources for reset:

- Power-on reset (POR)
- External pin reset (PIN)
- Computer operating properly (COP) timer
- Illegal opcode detect (IOP)
- Illegal address detect (ILAD)
- Low-voltage detect (LVD)
- Clock generator (MCG) loss of clock reset (LOC)
- Background debug forced reset

Each of these sources, except the background debug forced reset, has an associated bit in the system reset status register (SRS).

### 5.3.1 Computer Operating Properly (COP) Watchdog

The COP watchdog is intended to force a system reset when the application software fails to execute as expected. To prevent a system reset from the COP timer (when it is enabled), application software must reset the COP counter periodically. If the application program gets lost and fails to reset the COP counter before it times out, a system reset is generated to force the system back to a known starting point.

If the COP watchdog is not used in an application, it can be disabled by clearing SOPT1[COPT].

The COP counter is reset by writing 0x55 and 0xAA (in this order) to the address of SRS during the selected time-out period. Writes do not affect the data in the read-only SRS. When the write sequence is complete, the COP time-out period is restarted. If the program fails to do this during the time-out period, the microcontroller resets. See [Section 5.7.2, “System Reset Status Register \(SRS\),”](#) for additional information.

The SOPT2[COPCLKS] field (see [Section 5.7.4, “System Options 2 Register \(SOPT2\),”](#) for additional information) selects the clock source used for the COP timer. The clock source options are the bus clock or an internal 1 kHz clock source. With each clock source, there are three associated time-outs controlled by SOPT1[COPT]. [Table 5-7](#) summarizes the control functions of the COPCLKS and COPT bits. The COP watchdog defaults to operation from the 1 kHz clock source and the longest time-out ( $2^{10}$  cycles).

When the bus clock source is selected, windowed COP operation is available by setting SOPT1[COPW]. In this mode, writes to the SRS register to clear the COP timer must occur in the last 25% of the selected time-out period. A premature write immediately resets the microcontroller. When the 1 kHz clock source is selected, windowed COP operation is not available.

The COP counter is initialized by the first writes to the SOPT1 and SOPT2 registers and after any system reset. Subsequent writes to SOPT1 and SOPT2 have no effect on COP operation. This prevents accidental changes if the application program gets lost.

If the bus clock source is selected, the COP counter does not increment while the microcontroller is in background debug mode or while the system is in stop mode. The COP counter resumes when the microcontroller exits background debug mode or stop mode.

If the 1 kHz clock source is selected, the COP counter is re-initialized to zero after entry to either background debug mode or stop mode and begins from zero after exit from background debug mode or stop mode.

### 5.3.2 Illegal Opcode Detect (IOP)

By default the V1 ColdFire core enables the generation of an MCU reset in response to the processor's attempted execution of an illegal instruction (except for the ILLEGAL opcode), illegal line A, illegal line F instruction or the detection of a privilege violation (attempted execution of a supervisor instruction while in user mode).

The attempted execution of the STOP instruction with (SOPT[STOPE] and SOPT[WAITE] cleared) is treated as an illegal instruction.

The attempted execution of the HALT instruction with XCSR[ENBDM] cleared is treated as an illegal instruction.

The processor generates a reset in response to any of these events if CPUCR[IRD] is cleared. If this configuration bit is set, the processor generates the appropriate exception instead of forcing a reset.

### 5.3.3 Illegal Address Detect (ILAD)

By default the V1 ColdFire core enables the generation of an MCU reset in response to any processor-detected address error, bus error termination, RTE format error or fault-on-fault condition.

The processor generates a reset if CPUCR[ARD] is cleared. If this configuration bit is set, the processor generates the appropriate exception instead of forcing a reset, or simply halts the processor in response to the fault-on-fault condition.

## 5.4 Interrupts & Exceptions

The interrupt architecture of ColdFire utilizes 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 processor'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 processor requires that, once asserted, the interrupt source remain asserted until explicitly disabled by the interrupt service routine.

During the interrupt exception processing, the CPU does the following tasks in order:

1. enters supervisor mode,
2. disables trace mode,
3. uses the vector provided by the INTC when the interrupt was signaled (if CPUCR[IAE] is cleared) or explicitly fetches an 8-bit vector from the INTC (if CPUCR[IAE] is set).

This byte-sized operand fetch during exception processing is known as the interrupt acknowledge (IACK) cycle. The fetched data provides an index into the exception vector table which contains up to 256 addresses (depending upon the specific device), each pointing to the beginning of a specific exception service routine.

In particular, the first 64 exception vectors are reserved for the processor to handle reset, error conditions (access, address), arithmetic faults, system calls, etc. Vectors 64–255 are reserved for interrupt service routines. The MCF51CN128 series microcontrollers support 36 peripheral interrupt sources and an additional seven software interrupt sources. These are mapped into the standard seven ColdFire interrupt levels, with up to 9 levels of prioritization within a given level by the V1 ColdFire interrupt controller. See [Table 5-1](#) for details.

After the interrupt vector number has been retrieved, the processor continues by creating a stack frame in memory. For ColdFire, all exception stack frames are two 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. 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.

All ColdFire processors guarantee that the first instruction of the service routine is executed before interrupt sampling is resumed. By making this initial instruction a load of the SR, interrupts can be safely disabled, if required. Optionally, the processor can be configured to automatically raise the mask level to 7 for any interrupt during exception processing by setting CPUCR[IME].

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*. For additional information specific to this device, see [Chapter 8, “Interrupt Controller \(CF1\\_INTC\)](#).

## 5.4.1 **RESET/PTC3**

On the MCF51CN128 devices,  $\overline{\text{RESET}}$  is multiplexed with PTC3. This pin is open drain, and is also used during factory test for applying high voltages required for flash test purposes. Because of this, the  $\overline{\text{RESET}}$ /PTC3 pin is subject to the following restrictions:

1. This pin does not contain a clamp diode to  $V_{DD}$  and must not be driven above  $V_{DD}$ .
2. The voltage measured on the internally pulled up  $\overline{\text{RESET}}$  pin is not pulled to  $V_{DD}$ . The internal gates connected to this pin are pulled to  $V_{DD}$ . The  $\overline{\text{RESET}}$  pull-up must not be used to pull-up components external to the microcontroller.
3. On the MCF51CN128,  $\overline{\text{RESET}}$  co-exists on PTC3.  $\overline{\text{RESET}}$  is the default function upon power up. To avoid startup problems, PTC3 must only be programmed as an output function when used as GPIO.
4. Asserting a low on  $\overline{\text{RESET}}$ /PTC3, can wake the device from STOP2 mode, assuming that the pin is left programmed as  $\overline{\text{RESET}}$ .

## 5.4.2 **External Interrupt Request (IRQ) Pin**

External interrupts are managed by the IRQ status and control register, IRQSC. When the IRQ function is enabled, synchronous logic monitors the pin for edge-only or edge-and-level events. When the microcontroller is in stop mode and system clocks are shut down, a separate asynchronous path is used, so the IRQ pin (if enabled) can wake the microcontroller.

### 5.4.2.1 **Pin Configuration Options**

The IRQ pin enable (IRQPE) control bit in IRQSC must be set for the IRQ pin to act as the interrupt request (IRQ) input. As an IRQ input, you can choose the polarity of edges or levels detected (IRQEDG), whether the pin detects edges-only or edges and levels (IRQMOD), and whether an event causes an interrupt or only sets the IRQF flag which can be polled by software (IRQIE).

The IRQ pin, when enabled, defaults to use an internal pull device ( $\text{IRQPD} = 0$ ), configured as a pull-up or pull-down depending on the polarity chosen. To use an external pull-up or pull-down, the  $\text{IRQPD}$  can be set to turn off the internal device.

#### 5.4.2.2 Edge and Level Sensitivity

The  $\text{IRQMOD}$  control bit re-configures the detection logic so it detects edge events and pin levels. In the edge and level detection mode, the  $\text{IRQF}$  status flag sets when an edge is detected (when the IRQ pin changes from the deasserted to the asserted level), but the flag is continuously set (and cannot be cleared) if the IRQ pin remains at the asserted level.

#### 5.4.3 Interrupt Vectors, Sources, and Local Masks

**Table 5-1** shows address assignments for reset and interrupt vectors. The vector names shown in this table are the labels used in the Freescale Semiconductor-provided equate file for the MCF51CN128 series microcontrollers. The table is sorted by priority of the sources, with higher-priority sources at the top of the table. Note the highlighted entries which do not follow the address and vector number order of the surrounding vectors.

**Table 5-1. MC51CN128 Exception and Interrupt Vector Table**

Vector Address Offset	Interrupt Level	Priority	Vector Number	Stacked Program Counter	Vector Description	Enable	Source	Vector Name
0x000		N/A	0	—	Initial supervisor stack pointer	N/A	N/A	Vreset
0x004		N/A	1	—	Initial program counter	N/A	N/A	
0x008–0x0FC		N/A	2–63	—	Reserved for internal CPU exceptions (see Table 7-6)			
	7	7-4			Reserved			
0x100	7	mid	64	Next	IRQ_pin	IRQ_SC[IRQIE]	IRQ_SC[IRQF]	Virq
0x104	7	3	65	Next	Low_voltage_detect	PMC_LVDIE	PMC_LVDF	Vlvd
						PMC_LVWIE	PMC_LVWF	
0x108	7	2	66	Next	MCG_lock	MCG_C3[LOLIE]	MCG_SC[LOLS]	Vlol
	7	1			Reserved			
	6	7			RESERVED FOR REMAPPED VECTOR #1			VI6p7
	6	6			RESERVED FOR REMAPPED VECTOR #2			VI6p6
0x10C	6	5	67	Next	TPM1_ch0	TPM1_C0SC[CH0IE]	TPM1_C0SC[CH0F]	Vtpm1ch0
0x110	6	4	68	Next	TPM1_ch1	TPM1_C1SC[CH1IE]	TPM1_C1SC[CH1F]	Vtpm1ch1
0x114	6	3	69	Next	TPM1_ch2	TPM1_C2SC[CH1IE]	TPM1_C2SC[CH2F]	Vtpm1ch2

**Table 5-1. MC51CN128 Exception and Interrupt Vector Table (continued)**

<b>Vector Address Offset</b>	<b>Interrupt Level</b>	<b>Priority</b>	<b>Vector Number</b>	<b>Stacked Program Counter</b>	<b>Vector Description</b>	<b>Enable</b>	<b>Source</b>	<b>Vector Name</b>
0x118	6	2	70	Next	TPM1_ovfl	TPM1_SC[TOIE]	TPM1_SC[TOF]	Vtpm1ovf
0x11C	6	1	71	Next	MTIM1_ovfl	MTIM1_TOIE	MTIM1_TOF	
0x120	5	7	72	Next	TPM2_ch0	TPM2_C0SC[CH0IE]	TPM2_C0SC[CH0F ]	Vtpm2ch0
0x124	5	6	73	Next	TPM2_ch1	TPM2_C1SC[CH1IE]	TPM2_C1SC[CH1F ]	Vtpm2ch1
0x128	5	5	74	Next	TPM2_ch2	TPM2_C2SC[CH2IE]	TPM2_C2SC[CH2F ]	Vtpm2ch2
0x12C	5	4	75	Next	TPM2_ovfl	TPM2_SC[TOIE]	TPM2_SC[TOF]	Vtpm2ovf
0x130	5	3	76	Next	SPI1	SPI1_C1[SPIE]	SPI1_S[MODF] SPI1_S[SPRF]	Vspi1
						SPI1_C1[SPTIE]	SPI1_S[SPTEF]	
0x134	5	2	77	Next	SPI2	SPI2_C1[SPIE]	SPI2_S[MODF] SPI2_S[SPRF]	Vspi2
						SPI2_C1[SPTIE]	SPI2_S[SPTEF]	
0x138	5	1	78		MTIM2_ovfl	MTIM2_TOF	MTIM2_TOIE	
0x13C	4	7	79	Next	SCI1_err	SCI1_C3[ORIE]	SCI1_S1[OR]	Vsci1err
						SCI1_C3[FEIE]	SCI1_S1[FE]	
						SCI1_C3[NEIE]	SCI1_S1[NF]	
						SCI1_C3[PEIE]	SCI1_S1[PF]	
0x140	4	6	80	Next	SCI1_rx	SCI1_C2[RIE]	SCI1_S1[RDRF]	Vsci1rx
						SCI1_C2[ILIE]	SCI1_S1[IDLE]	
						SCI1_BDH[LBKDIIE]	SCI1_S2[LBKDIF]	
						SCI1_BDH[RXEDGIE]	SCI1_S2[RXEDGIF ]	
0x144	4	5	81	Next	SCI1_tx	SCI1_C2[TCIE]	SCI1_S1[TC]	Vsci1tx
						SCI1_C2[TIE]	SCI1_S1[TDRE]	
0x148	4	4	82	Next	SCI2_err	SCI2_C3[ORIE]	SCI2_S1[OR]	Vsci2err
						SCI2_C3[FEIE]	SCI2_S1[FE]	
						SCI2_C3[NEIE]	SCI2_S1[NF]	
						SCI2_C3[PEIE]	SCI2_S1[PF]	

**Table 5-1. MC51CN128 Exception and Interrupt Vector Table (continued)**

Vector Address Offset	Interrupt Level	Priority	Vector Number	Stacked Program Counter	Vector Description	Enable	Source	Vector Name
0x14C	4	3	83	Next	SCI2_rx	SCI2_C2[RIE]	SCI2_S1[RDRF]	Vsci2rx
						SCI2_C2[ILIE]	SCI2_S1[IDLE]	
						SCI2_BDH[LBKDIIE]	SCI2_S2[LBKDIF]	
						SCI2_BDH[RXEDGIE]	SCI2_S2[RXEDGIF]	
0x150	4	2	84	Next	SCI2_tx	SCI2_C2[TCIE]	SCI2_S1[TC]	Vsci2tx
						SCI2_C2[TIE]	SCI2_S1[TDRE]	
0x154	4	1	85	Next	SCI3_OR <sup>1</sup>	See Vectors 100 - 102		Vsci3or
0x158	3	7	86	Next	FEC TXF	FEC_EIMR[TXF]	FEC_EIR[TXF]	
0x15C	3	6	87	Next	FEC RXF	FEC_EIMR[RXF]	FEC_EIR[RXF]	
0x160	3	5	88	Next	FEC Other <sup>2</sup>	See Vectors 88 - 98		
0x164	3	4	89	Next	FEC HBERR	FEC_EIMR[HBERR]	FEC_EIR[HBERR]	
0x168	3	3	90	Next	FEC BABR	FEC_EIMR[BABR]	FEC_EIR[BABR]	
0x16C	3	2	91	Next	FEC BABT	FEC_EIMR[BABT]	FEC_EIR[BABT]	
0x170	3	1	92	Next	FEC GRA	FEC_EIMR[GRA]	FEC_EIR[GRA]	
0x174	2	7	93	Next	FEC TXB	FEC_EIMR[TXB]	FEC_EIR[TXB]	
0x178	2	6	94	Next	FEC RXB	FEC_EIMR[RXB]	FEC_EIR[RXB]	
0x17C	2	5	95	Next	FEC MII	FEC_EIMR[MII]	FEC_EIR[MII]	
0x180	2	4	96	Next	FEC EBERR	FEC_EIMR[EBERR]	FEC_EIR[EBERR]	
0x184	2	3	97	Next	FEC LC	FEC_EIMR[LC]	FEC_EIR[LC]	
0x188	2	2	98	Next	FEC RL	FEC_EIMR[RL]	FEC_EIR[RL]	
0x18C	2	1	99	Next	FEC UN	FEC_EIMR[UN]	FEC_EIR[UN]	
0x190	1	7	100	Next	SCI3_err	SCI3_C3[ORIE]	SCI3_S1[OR]	Vsci3err
						SCI3_C3[FEIE]	SCI3_S1[FE]	
						SCI3_C3[NEIE]	SCI3_S1[NF]	
						SCI3_C3[PEIE]	SCI3_S1[PF]	
0x194	1	6	101	Next	SCI3_rx	SCI3_C2[RIE]	SCI3_S1[RDRF]	Vsci3rx
						SCI3_C2[ILIE]	SCI3_S1[IDLE]	
						SCI3_BDH[LBKDIIE]	SCI3_S2[LBKDIF]	
						SCI3_BDH[RXEDGIE]	SCI3_S2[RXEDGIF]	

**Table 5-1. MC51CN128 Exception and Interrupt Vector Table (continued)**

Vector Address Offset	Interrupt Level	Priority	Vector Number	Stacked Program Counter	Vector Description	Enable	Source	Vector Name
0x198	1	5	102	Next	SCI3_tx	SCI3_C2[TCIE]	SCI3_S1[TC]	Vsci3tx
						SCI3_C2[TIE]	SCI3_S1[TDRE]	
0x19C	7	0	103	Next	Level 7 Software Interrupt			Force_Lvl7
0x1A0	6	0	104	Next	Level 6 Software Interrupt			Force_Lvl6
0x1A4	5	0	105	Next	Level 5 Software Interrupt			Force_Lvl5
0x1A8	4	0	106	Next	Level 4 Software Interrupt			Force_Lvl4
0x1AC	3	0	107	Next	Level 3 Software Interrupt			Force_Lvl3
0x1B0	2	0	108	Next	Level 2 Software Interrupt			Force_Lvl2
0x1B4	1	0	109	Next	Level 1 Software Interrupt			Force_Lvl1
0x1B8	1	4	110	Next	IIC1	IIC1_CR1[IICIE]	IIC1_SR[IICIF]	Viic1
0x1BC	1	mid	111	Next	IIC2	IIC2_CR1[IICIE]	IIC2_SR[IICIF]	Viic2
0x1C0	1	3	112	Next	ADC1	ADC_SC1[AIEN]	ADC_SC1[COCO]	Vadc
0x1C4	1	2	113	Next	KBI1 and KBI2	KBI1_SC[KBIE] KBI2_SC[KBIE]	KBI1_SC[KBF] KBI2_SC[KBF]	Vkeyboard
0x1C8	1	1	114	Next	RTC	RTC_SC[RTIE]	RTC_SC[RTIF]	Vrtc
0x1CC – 0x3FC	N/A	N/A	115 – 255	Next	Reserved			

- <sup>1</sup> The SCI3\_OR interrupt is a logical ORing of SCI3\_rx, SCI3\_tx and SCI3\_err. You can use this vector for SCI3 interrupts to be processed at priority level 4. Use of SCI3\_OR is mutually exclusive with use of the other SCI3 vectors.
- <sup>2</sup> The “FEC Other” interrupt is a logical ORing of the FEC HBERR, BABR, BABT, GRA, TXB, RXB, MII, EBERR, LC, RL and UN interrupts. You can use this vector to assign all FEC interrupts at priority level 3. Use of “FEC Other” is mutually exclusive with the use of the interrupt signals used to generate it.

**Table 5-2. Unsupported FTSR Interrupts**

Interrupt	FTSR_Local Enable	FTSR_Source	Description
FTSR_emptybuf	FTSR_FCNFG[CBEIE]	FTSR_FSTAT[FCBEF]	Flash Command Buffer Empty Interrupt
FTSR_coco	FTSR_FCNFG[CCIE]	FTSR_FSTAT[FCCF]	Flash Command Complete Interrupt

Vector numbers which are not shown in [Table 5-1](#) are the standard set of ColdFire exceptions, many of which apply to this device. These are listed below in [Table 5-3](#).

Additional details with regard to interrupt processing are present in Chapter 8, “[Interrupt Controller \(CF1\\_INTC\)](#).”

The CPU configuration register (CPUCR) within the supervisor programming model allows you to determine if specific ColdFire exception conditions are to generate a normal exception or a system reset. The default state of the CPUCR forces a system reset for any of the exception types listed in [Table 5-3](#).

**Table 5-3. ColdFire Exception Vector Table<sup>1</sup>**

Vector Number	Exception	Reset Disabled via CPUCR	Reported using SRS
64-108	I/O Interrupts	N/A	—
61	Unsupported instruction	N/A	—
47	Trap #15	N/A	—
46	Trap #14	N/A	—
45	Trap #13	N/A	—
44	Trap #12	N/A	—
43	Trap #11	N/A	—
42	Trap #10	N/A	—
41	Trap #9	N/A	—
40	Trap #8	N/A	—
39	Trap #7	N/A	—
38	Trap #6	N/A	—
37	Trap #5	N/A	—
36	Trap #4	N/A	—
35	Trap #3	N/A	—
34	Trap #2	N/A	—
33	Trap #1	N/A	—
32	Trap #0	N/A	—
24	Spurious IRQ	N/A	—
14	Format error	CPUCR[31]	ilad
12	Debug breakpoint IRQ	N/A	—
11	Illegal LineF	CPUCR[30]	ilop
10	Illegal LineA	CPUCR[30]	ilop
9	Trace	N/A	—
8	Privileged Violation	CPUCR[30]	ilop
4	Illegal instruction	CPUCR[30]	ilop <sup>2</sup>
3	Address error	CPUCR[31]	ilad
2	Access error	CPUCR[31]	ilad
n/a	Flt-on-Flt Halt	CPUCR[31]	ilad

- <sup>1</sup> Exception vector numbers not appearing in this table are not applicable to the V1 core and are reserved.
- <sup>2</sup> The execution of the ILLEGAL instruction (0x4AFC) always generates an illegal instruction exception, regardless of the state of CPUCR[30].

## 5.5 Low-Voltage Detect (LVD) System

The MCF51CN128 series microcontroller includes a system to protect against low voltage conditions to protect memory contents and control microcontroller system states during supply voltage variations. The system is comprised of a power-on reset (POR) circuit and a LVD circuit with a user-selectable trip voltage, high ( $V_{LVDH}$ ) or low ( $V_{LVDL}$ ). The LVD circuit is enabled when the SPMSC1[LVDE] bit is set and the trip voltage is selected by the SPMSC3[LVDV] bit. If LVDE and LVDSE are set when the STOP instruction is processed, the device enters STOP4 mode. The LVD can be left enabled in this mode.

### 5.5.1 Power-On Reset Operation

When power is initially applied to the microcontroller, or when the supply voltage drops below the power-on reset re-arm voltage level,  $V_{POR}$ , the POR circuit causes a reset condition. As the supply voltage rises, the LVD circuit holds the microcontroller in reset until the supply has risen above the LVD low threshold,  $V_{LVDL}$ . Both the POR bit and the LVD bit in SRS are set following a POR.

### 5.5.2 LVD Reset Operation

The LVD can be configured to generate a reset upon detection of a low voltage condition by setting LVDRE. The low voltage detection threshold is determined by the LVDV bit. After an LVD reset has occurred, the LVD system holds the microcontroller in reset until the supply voltage has risen above the low voltage detection threshold. The SRS[LVD] bit is set following an LVD reset or POR.

### 5.5.3 LVD Interrupt Operation

When a low voltage condition is detected and the LVD circuit is configured using SPMSC1 for interrupt operation (LVDE set, LVDIE set, and LVDRE clear), then SPMSC1[LVDF] is set and an LVD interrupt request occurs. The LVDF bit is cleared by writing a 1 to SPMSC1[LVDACK].

### 5.5.4 Low-Voltage Warning (LVW) Interrupt Operation

The LVD system has a low voltage warning flag (LVWF) to indicate that the supply voltage is approaching, but is above, the LVD voltage. The LVW also has an interrupt associated with it, enabled by setting the SPMSC3[LVWIE] bit. If enabled, an LVW interrupt request occurs when the LVWF is set. LVWF is cleared by writing a 1 to the SPMSC3[LVWACK] bit. There are two user-selectable trip voltages for the LVW, one high ( $V_{LVWH}$ ) and one low ( $V_{LVWL}$ ). The trip voltage is selected by SPMSC3[LVWV] bit.

## 5.6 Peripheral Clock Gating

The MCF51CN128 series microcontroller includes a clock gating system to manage the bus clock sources to the individual peripherals. Using this system, you can enable or disable the bus clock to each of the peripherals at the clock source, eliminating unnecessary clocks to peripherals which are not in use; thereby reducing the overall run and wait mode currents.

Out of reset, all peripheral clocks are enabled. For lowest possible run or wait currents, you should disable the clock source to any peripheral not in use. The actual clock is enabled or disabled immediately following the write to the clock gating control registers (SCGC1-4). Any peripheral with a gated clock can not be used unless its clock is enabled. Writing to the registers of a peripheral with a disabled clock (other than FEC and Mini-FlexBus) has no effect. When the FEC and Mini-FlexBus are not available or their clocks are disabled, any attempt to read or write the FEC and Mini-Bus register bits results in an access error.

### NOTE

User software must disable the peripheral before disabling the clocks to the peripheral. When clocks are re-enabled to a peripheral, the peripheral registers need to be re-initialized by user software.

In stop modes, the bus clock is disabled for all gated peripherals, regardless of the settings in the SCGC1, SCGC2, SCGC3, and SCGC4 registers.

## 5.7 Reset, Interrupt, and System Control Registers and Control Bits

One 8-bit register in the direct page register space and eight 8-bit registers in the register space are related to reset and interrupt systems.

Refer to [Section 4.2, “Detailed Register Addresses and Bit Assignments,”](#) for the absolute address assignments for all registers. This section refers to registers and control bits only by their names. A Freescale-provided equate or header file is used to translate these names into the appropriate absolute addresses.

Some control bits in the SOPT1 and SPMSC2 registers are related to modes of operation. Although brief descriptions of these bits are provided here, the related functions are discussed in greater detail in [Chapter 3, “Modes of Operation.”](#)

### 5.7.1 Interrupt Pin Request Status and Control Register (IRQSC)

This direct page register includes status and control bits which configure the IRQ function, report status, and acknowledge IRQ events.

	7	6	5	4	3	2	1	0
R	0	IRQPDD	IRQEDG	IRQPE	IRQF	0	IRQIE	IRQMOD
W						IRQACK		
Reset	0	0	0	0	0	0	0	0

Figure 5-1. Interrupt Request Status and Control Register (IRQSC)

**Table 5-4. IRQSC Register Field Descriptions**

Field	Description
7	Reserved, must be cleared.
6 IRQPDD	<b>Interrupt Request (IRQ) Pull Device Disable</b> — This read/write control bit disables the internal pull-up/pull-down device when the IRQ pin is enabled (IRQPE = 1) allowing for an external device to be used. 0 IRQ pull device is enabled if IRQPE = 1. 1 IRQ pull device is disabled if IRQPE = 1. This bit overrides the pull-up enable logic in the GPIO controls when IRQ is assigned this pin.
5 IRQEDG	<b>Interrupt Request (IRQ) Edge Select</b> — This read/write control bit selects the polarity of edges or levels on the IRQ pin that causes IRQF to set. The IRQMOD control bit determines whether the IRQ pin is sensitive to both edges and levels or only edges. When IRQEDG is set and the internal pull device is enabled, the pull-up device is reconfigured as an optional pull-down device. 0 IRQ is falling edge or falling edge/low-level sensitive. 1 IRQ is rising edge or rising edge/high-level sensitive.
4 IRQPE	<b>IRQ Pin Enable</b> — This read/write control bit enables the IRQ pin function. When this bit is set, the IRQ pin can be used as an external interrupt request. Note that the IRQ pin function must also be enabled via the MC <sup>1</sup> registers. 0 IRQ pin function is disabled. 1 IRQ pin function is enabled.
3 IRQF	<b>IRQ Flag</b> — This read-only status bit indicates when an interrupt request event has occurred. 0 No IRQ request. 1 IRQ event detected.
2 IRQACK	<b>IRQ Acknowledge</b> — This write-only bit is used to acknowledge interrupt request events (write 1 to clear IRQF). Writing 0 has no meaning or effect. Reads always return 0. If edge-and-level detection is selected (IRQMOD = 1), IRQF cannot be cleared while the IRQ pin remains at its asserted level.
1 IRQIE	<b>IRQ Interrupt Enable</b> — This read/write control bit determines whether IRQ events generate an interrupt request. 0 Interrupt request when IRQF set is disabled (use polling). 1 Interrupt requested whenever IRQF is set.
0 IRQMOD	<b>IRQ Detection Mode</b> — This read/write control bit selects edge-only detection or edge-and-level detection. The IRQEDG control bit determines the polarity of edges and levels that are detected as interrupt request events. See <a href="#">Section 5.4.2.2, “Edge and Level Sensitivity,”</a> for more details. 0 IRQ event on falling edges or rising edges only. 1 IRQ event on falling edges and low levels or on rising edges and high levels.

<sup>1</sup> MC = Port Mux Control

## 5.7.2 System Reset Status Register (SRS)

This register includes read-only status flags to indicate the source of the most recent reset. When a debug host forces reset by setting CSR2[BDFR], none of the status bits in SRS are set. Writing any value to this register address clears the COP watchdog timer without affecting the contents of this register. The reset state of these bits depends on what caused the microcontroller to reset.

	7	6	5	4	3	2	1	0
R	POR	PIN	COP	ILOP	ILAD	LOC	LVD	0
W	Writing any value to SRS address clears COP watchdog timer.							
POR:	1	0	0	0	0	0	1	0
LVD:	U	0	0	0	0	0	1	0
Any other reset:	0	-1	-1	-1	-1	0	0	0

<sup>1</sup> Any of these reset sources that are active at the time of reset entry causes the corresponding bit to set; bits corresponding to sources that are not active at the time of reset entry are cleared.

<sup>2</sup> U = Unaffected by MCU Reset.

Figure 5-2. System Reset Status (SRS)

Table 5-5. SRS Register Field Descriptions

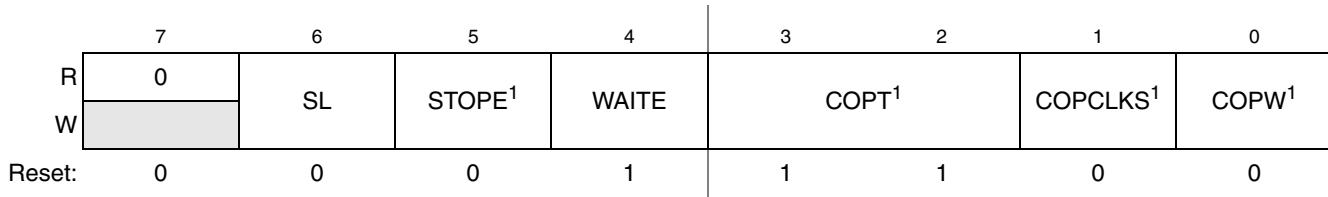
Field	Description
7 POR	<b>Power-On Reset</b> — Reset was caused by the power-on detection logic. Because the internal supply voltage was ramping up at the time, the low-voltage reset (LVD) status bit is also set to indicate that the reset occurred while the internal supply was below the LVD threshold. 0 Reset not caused by POR. 1 POR caused reset.
6 PIN	<b>External Reset Pin</b> — Reset was caused by an active-low level on the external reset pin. 0 Reset not caused by external reset pin. 1 Reset came from external reset pin.
5 COP	<b>Computer Operating Properly (COP) Watchdog</b> — Reset was caused by the COP watchdog timer timing out. This reset source can be blocked by clearing SOPT1[COPT]. 0 Reset not caused by COP time-out. 1 Reset caused by COP time-out.
4 ILOP	<b>Illegal Opcode</b> — Reset was caused by an attempt to execute an unimplemented or illegal opcode. This includes any illegal instruction [except the ILLEGAL (0x4AFC) opcode] or a privilege violation (execution of a privileged instruction in user mode). The STOP instruction is considered illegal if stop is disabled by both SOPT[STOPE, WAITE] being cleared. The HALT instruction is considered illegal if the BDM interface is disabled by clearing XCSR[ENBDM]. 0 Reset not caused by an illegal opcode. 1 Reset caused by an illegal opcode.
3 ILAD	<b>Illegal Address</b> — Reset was caused by the processor's attempted access of an illegal address in the memory map, an address error, an RTE format error or the fault-on-fault condition. All the illegal address resets are enabled when CPUCR[ARD] is cleared. When CPUCR[ARD] is set, then the appropriate processor exception is generated instead of the reset, or if a fault-on-fault condition is reached, the processor simply halts. 0 Reset not caused by an illegal access. 1 Reset caused by an illegal access.

**Table 5-5. SRS Register Field Descriptions (continued)**

Field	Description
2 LOC	<b>Loss-of-Clock Reset</b> — Reset was caused by a loss of external clock. MCGC3[CME] must be set for this function to operate. 0 Reset not caused by a loss of external clock. 1 Reset caused by a loss of external clock.
1 LVD	<b>Low Voltage Detect</b> — If the LVD is enabled with LVDRE set, and the supply drops below the LVD trip voltage, an LVD reset occurs. This bit is also set by POR. 0 Reset not caused by LVD trip or POR. 1 Reset caused by LVD trip or POR.

### 5.7.3 System Options 1 Register (SOPT1)

This register has three write-once bits and one write anytime bit. For the write-once bits, only the first write after reset is honored. All bits in the register can be read at any time. Any subsequent attempt to write a write-once bit is ignored to avoid accidental changes to these sensitive settings. SOPT1 must be written during the reset initialization program to set the desired controls, even if the desired settings are the same as the reset settings.



<sup>1</sup> These bits can be written only one time after reset. Subsequent writes are ignored.

**Figure 5-3. System Options 1 Register (SOPT1)****Table 5-6. SOPT1 Field Descriptions**

Field	Description
7	Reserved, must be cleared.
6 SL	<b>Security Level</b> — If security is enabled by the mechanisms outlined in <a href="#">Section 4.5, “Security”</a> , then this bit affects what CPU operations can access off-chip by the Mini-FlexBus interface. This bit has no effect if security is not enabled. 0 All off-chip accesses (opcode and data) by the Mini-FlexBus are disallowed. 1 Off-chip opcode accesses are disallowed. Data accesses are allowed.
5 STOPE	<b>Stop Mode Enable</b> — This write-once bit is used to enable stop mode. If stop and wait modes are disabled and a user program attempts to execute a STOP instruction, an illegal opcode reset may be generated depending on CPUCR[IRD].
4 WAITE	<b>WAIT Mode Enable</b> — This write-anytime bit is used to enable wait mode. If stop and wait modes are disabled and a user program attempts to execute a STOP instruction, an illegal opcode reset may be generated depending on CPUCR[IRD].
3-2 COPT	<b>COP Watchdog Time-out</b> — These write-once bits select the time-out period of the COP. COPT along with SOPT2[COPCLKS] defines the COP time-out period as described in <a href="#">Table 5-7</a> .

**Table 5-6. SOPT1 Field Descriptions (continued)**

Field	Description
1 COPCLKS	<b>COP Watchdog Clock Select</b> — This write-once bit selects the clock source of the COP watchdog. 0 Internal 1 kHz clock is source to COP. 1 Bus clock is source to COP.
0 COPW	<b>COP Window Mode</b> — This write-once bit specifies whether the COP operates in normal or window mode. In Window mode, the 0x55–0xAA write sequence to the SRS register must occur within the last 25% of the selected period; any write to the SRS register during the first 75% of the selected period resets the microcontroller. 0 Normal mode 1 Window mode

**NOTE**

If STOPE and WAITE bits are set, the WAITE bit has precedence.

**Table 5-7. COP Configuration Options**

Control Bits		Clock Source	COP Window <sup>1</sup> Opens (SOPT2[COPW] = 1)	COP Overflow Count
SOPT1[COPCLKS]	SOPT1[COPT]			
N/A	00	N/A	N/A	COP is disabled
0	01	1 kHz	N/A	$2^5$ cycles (32 ms <sup>2</sup> )
0	10	1 kHz	N/A	$2^8$ cycles (256 ms <sup>1</sup> )
0	11	1 kHz	N/A	$2^{10}$ cycles (1,024 ms <sup>1</sup> )
1	01	Bus	6,144 cycles	$2^{13}$ cycles
1	10	Bus	49,152 cycles	$2^{16}$ cycles
1	11	Bus	196,608 cycles	$2^{18}$ cycles

<sup>1</sup> Windowed COP operation requires you to clear the COP timer in the last 25% of the selected time-out period. This column displays the minimum number of clock counts required before the COP timer can be reset when in windowed COP mode (SOPT2[COPW] = 1).

<sup>2</sup> Values shown in milliseconds based on t<sub>LPO</sub> = 1 ms.

## 5.7.4 System Options 2 Register (SOPT2)

This register contains bits that control the PMC LVD trim. This is a reserved register and must not be written by application code.

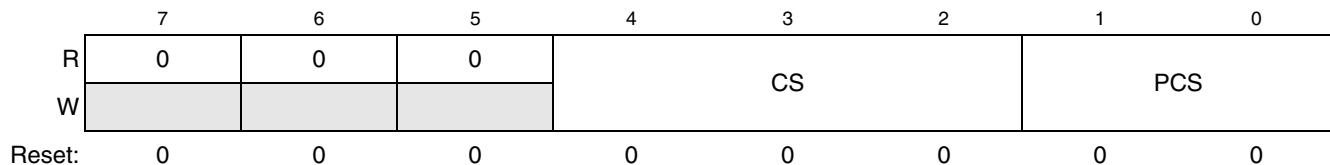
**Figure 5-4. System Options 2 Register (SOPT2)**

**Table 5-8. SOPT2 Field Descriptions**

Field	Description
7 RSVD	Reserved, must be cleared.
6–5 FC	<b>Flash Configuration</b> — These bits specify the amount of flash memory available on this device. 00 64 KB Memory Map 01 Reserved 10 96 KB Memory Map 11 128 KB Memory Map
4–0 PMC_LVD_TRIM	Reserved, must be cleared.

## 5.7.5 System Options Register 3 (SOPT3)

This register controls the output mux functions of the PHYCLK and CLKOUT pins. The various clock sources must be enabled/disabled by the appropriate controls elsewhere in the device.

**Figure 5-5. System Options Register 3 (SOPT3)****Table 5-9. SOPT3 Field Descriptions**

Field	Description
7–5	Reserved, must be cleared.
4–2 CS	<b>CLKOUT Select</b> 000 Disabled 001 OSCOUT 010 MCGOUT 011 BUSCLK 100 LPOCLK Others Reserved
1–0 PCS	<b>Phy Clock Select</b> 00 Disabled 01 OSCOUT 10 MCGOUT 11 BUSCLK

## 5.7.6 System Device Identification Register (SDIDH, SDIDL)

These read-only registers identify the ColdFire derivative. This allows the development software to recognize where specific memory blocks, registers, and control bits are located in a target microcontroller.

Additional configuration information about the ColdFire core and memory system is loaded into the 32-bit D0 (core) and D1 (memory) registers at reset. This information can be stored into memory by the system startup code for later use by configuration-sensitive application code. See [Section 7.3.3.14, “Reset Exception,”](#) for more information.

	7	6	5	4		3	2	1	0
R	REV <sup>1</sup>				ID11	ID10	ID9	ID8	
W									
Reset:	-	-	-	-	1	1	0	0	

**Figure 5-6. System Device Identification Register — High (SDIDH)**

<sup>1</sup> The reset value of 4-bit REV is 0000 for 1.0 Silicon and 0001 for 1.1 Silicon.

**Table 5-10. SDIDH Register Field Descriptions**

Field	Description
7–4 REV	<b>Revision Number</b> — This field indicates the chip revision number.
3–0 ID[11:8]	<b>Part Identification Number</b> — Each derivative in the ColdFire family has a unique identification number. The MCF51CN128 series microcontrollers have these bits hard coded to the value 0xC. See also ID bits in <a href="#">Table 5-11</a> . The overall device ID is 0xC2F.

	7	6	5	4		3	2	1	0
R	ID7	ID6	ID5	ID4	ID3	ID2	ID1	ID0	
W									
Reset:	0	0	1	0	1	1	1	1	1

**Figure 5-7. System Device Identification Register — Low (SDIDL)**

**Table 5-11. SDIDL Register Field Descriptions**

Field	Description
7–0 ID[7:0]	<b>Part Identification Number</b> — Each derivative in the ColdFire family has a unique identification number. The MCF51CN128 series microcontrollers have these bits hard coded to the value 0x2F. The overall device ID is 0xC2F.

## 5.7.7 System Power Management Status and Control 1 Register (SPMSC1)

This register contains status and control bits to support the low voltage detect function, and to enable the bandgap voltage reference for the ADC module. This register must be written during the reset initialization program to set the desired controls even if the desired settings are the same as the reset settings.

	7	6	5	4		3	2	1	0
R	LVDF <sup>1</sup>	0	LVDIE	LVDRE <sup>2</sup>	LVDSE	LVDE <sup>2</sup>	0	BGBE	
		LVDACK							
Reset:	0	0	0	1	1	1	0	0	0

<sup>1</sup> LVDF is set when V<sub>Supply</sub> transitions below the trip point or after reset and V<sub>Supply</sub> is already below V<sub>LVW</sub>.

<sup>2</sup> This bit can be written only one time after reset. Additional writes are ignored.

**Figure 5-8. System Power Management Status and Control 1 Register (SPMSC1)**

**Table 5-12. SPMSC1 Register Field Descriptions**

Field	Description
7 LVDF	<b>Low-Voltage Detect Flag</b> — The LVDF bit indicates the low-voltage detect event status. 0 Low-voltage warning is not present. 1 Low-voltage warning is present or was present.
6 LVDACK	<b>Low-Voltage Detect Acknowledge</b> — If LVDF is set, a low-voltage condition has occurred. To acknowledge this low-voltage detection, write 1 to LVDACK, which automatically clears LVDF if the low-voltage detection is no longer present.
5 LVDIE	<b>Low-Voltage Detect Interrupt Enable</b> — This bit enables hardware interrupt requests for LVDF. 0 Hardware interrupt disabled (use polling). 1 Request a hardware interrupt when LVDF is set.
4 LVDRE	<b>Low-Voltage Detect Reset Enable</b> — This write-once bit enables LVDF events to generate a hardware reset (if LVDE is set). 0 LVDF does not generate hardware resets. 1 Force an MCU reset when an enabled low-voltage detect event occurs.
3 LVDSE	<b>Low-Voltage Detect Stop Enable</b> — If LVDE is set, this read/write bit determines if the low-voltage detect function operates when the MCU is in stop mode. 0 Low-voltage detect disabled during stop mode. 1 Low-voltage detect enabled during stop mode.
2 LVDE	<b>Low-Voltage Detect Enable</b> — This write-once bit enables low-voltage detect logic and qualifies the operation of other bits in this register. 0 LVD logic disabled. 1 LVD logic enabled.
0 BGBE	<b>Bandgap Buffer Enable</b> — This bit enables an internal buffer for the bandgap voltage reference for use by the ADC module on one of its internal channels. 0 Bandgap buffer disabled. 1 Bandgap buffer enabled.

## 5.7.8 System Power Management Status and Control 2 Register (SPMSC2)

This register contains status and control bits to configure the low power run and wait modes and configure the stop mode behavior of the microcontroller. See [Section 3.7.2, “Low-Power Wait Mode \(LPwait\),”](#) for more information.

SPMSC2 is not reset when exiting from STOP2.

	7	6	5	4	3	2	1	0
R	LPR	LPRS	LPWUI	0	PPDF	0	PPDE <sup>1</sup>	PPDC
W				0		PPDACK		
Reset:	0	0	0	0	-	0	1	0

**Figure 5-9. System Power Management Status and Control 2 Register (SPMSC2)**

<sup>1</sup> PPDE is a write-once bit that can be used to permanently disable the PPDC bit.

**Table 5-13. SPMSC2 Register Field Descriptions**

Field	Description
7 LPR	<b>Low-Power Regulator Control</b> — The LPR bit controls entry into the low-power run and low-power wait modes in which the voltage regulator is put into standby. This bit cannot be set if PPDC is set. If PPDC and LPR are set in a single write instruction, only PPDC is actually set. LPR is cleared when an interrupt occurs in low-power mode and the LPWUI bit is set. 0 Low-power run and low-power wait modes are disabled. 1 Low-power run and low-power wait modes are requested.
6 LPRS	<b>Low-Power Regulator Status</b> — This read-only status bit indicates that the voltage regulator has entered into standby for the low-power run or wait mode. 0 The voltage regulator is not currently in standby. 1 The voltage regulator is currently in standby.
5 LPWUI	<b>Low-Power Wake-up on Interrupt</b> — This bit controls if the voltage regulator exits standby when any active MCU interrupt occurs. 0 The voltage regulator remains in standby on an interrupt. 1 The voltage regulator exits standby on an interrupt. LPR is cleared.
4	Reserved, must be cleared.
3 PPDF	<b>Partial Power-Down Flag</b> — This read-only status bit indicates that the microcontroller has recovered from stop2 mode. 0 Microcontroller has not recovered from stop2 mode. 1 Microcontroller recovered from stop2 mode.
2 PPDACK	<b>Partial Power-Down Acknowledge</b> — Writing 1 to PPDACK clears the PPDF bit.
1 PPDE	<b>Partial Power-Down Enable</b> — The write-once PPDE bit locks the partial power-down feature. This is a write-once bit. 0 Partial power-down is not enabled. 1 Partial power-down is enabled and controlled by the PPDC bit.
0 PPDC	<b>Partial Power-Down Control</b> — The PPDC bit controls which power-down mode is selected. This bit cannot be set if LPR is set. If PPDC and LPR are set in a single write instruction, only PPDC is actually set. PPDE must be set for PPDC to be set. 0 Stop3 low power mode enabled. 1 Stop2 partial power-down mode enabled.

## 5.7.9 System Power Management Status and Control 3 Register (SPMSC3)

This register reports the status of the low voltage warning function and selects the low voltage detect trip voltage. SPMSC3 is not reset when exiting from stop2.

	7	6	5	4	3	2	1	0
R	LVWF	0	LVDV	LVWV	LVWIE	0	0	0
W		LVWACK						
POR:	0 <sup>1</sup>	0	0	0	0	0	0	0
LVR:	0 <sup>1</sup>	0	U	U	0	0	0	0
Any other reset:	0 <sup>1</sup>	0	U	U	0	0	0	0

<sup>1</sup> LVWF is set when V<sub>Supply</sub> transitions below the trip point or after reset and V<sub>Supply</sub> is already below V<sub>LVW</sub>.

<sup>2</sup> U = Unaffected by MCU Reset.

**Figure 5-10. System Power Management Status and Control 3 Register (SPMSC3)**

**Table 5-14. SPMSC3 Register Field Descriptions**

Field	Description
7 LVWF	<b>Low-Voltage Warning Flag</b> — The LVWF bit indicates the low voltage detect event status. 0 Low voltage warning not present. 1 Low voltage warning is present or was present.
6 LVWACK	<b>Low-Voltage Warning Acknowledge</b> — Writing a 1 to LVWACK clears LVWF if a low voltage warning is no longer present.
5 LVDV	<b>Low-Voltage Detect Voltage Select</b> — The LVDV bit selects the LVD trip point voltage (V <sub>LVD</sub> ). 0 Low trip point selected (V <sub>LVD</sub> = V <sub>LVDL</sub> ). 1 High trip point selected (V <sub>LVD</sub> = V <sub>LVDH</sub> ).
4 LVWV	<b>Low-Voltage Warning Voltage Select</b> — The LVWV bit selects the LVW trip point voltage (V <sub>LVW</sub> ). 0 Low trip point selected (V <sub>LVW</sub> = V <sub>LVWL</sub> ). 1 High trip point selected (V <sub>LVW</sub> = V <sub>LVWH</sub> ).
3 LVWIE	<b>Low-Voltage Warning Interrupt Enable</b> — This bit enables hardware interrupt requests for LVWF. 0 Hardware interrupt disabled (use polling). 1 Request a hardware interrupt when LVWF is set.
2–0	Reserved, must be cleared.

**Table 5-15. LVD and LVW Trip Point Typical Values<sup>1</sup>**

LVDV:LVWV	LVW Trip Point	LVD Trip Point
00	V <sub>LVWL</sub> = 2.16	V <sub>LVDL</sub> = 1.84
01	V <sub>LVWL</sub> = 2.21	V <sub>LVDL</sub> = 1.92
10 Not Recommended	V <sub>LVWL</sub> = 2.46	V <sub>LVDL</sub> = 2.33
11		

<sup>1</sup> See the *MCF51CN128 Data Sheet* for minimum and maximum values.

## 5.7.10 System Clock Gating Control 1 Register (SCGC1)

This register contains control bits to enable or disable the bus clock to three of the timers (MTIM2 is controlled by SCGC4), ADC, IICs and two of the three SCI modules on the chip. Gating off the clocks to unused peripherals reduces the microcontroller's run and wait currents. See [Section 5.6, “Peripheral Clock Gating,”](#) for more information.

	7	6	5	4	3	2	1	0
R W	MTIM1	TPM2	TPM1	ADC	IIC2	IIC1	SCI2	SCI1
Reset:	1	1	1	1	1	1	1	1

Figure 5-11. System Clock Gating Control 1 Register (SCGC1)

Table 5-16. SCGC1 Register Field Descriptions

Field	Description
7 MTIM1	<b>MTIM1 Clock Gate Control</b> 0 Bus clock to the MTIM1 module is disabled. 1 Bus clock to the MTIM1 module is enabled.
6 TPM2	<b>TPM2 Clock Gate Control</b> 0 Bus clock to the TPM2 module is disabled. 1 Bus clock to the TPM2 module is enabled.
5 TPM1	<b>TPM1 Clock Gate Control</b> 0 Bus clock to the TPM1 module is disabled. 1 Bus clock to the TPM1 module is enabled.
4 ADC	<b>ADC Clock Gate Control</b> 0 Bus clock to the ADC module is disabled. 1 Bus clock to the ADC module is enabled.
3 IIC2	<b>IIC2 Clock Gate Control.</b> 0 Bus clock to the IIC2 module is disabled. 1 Bus clock to the IIC2 module is enabled.
2 IIC1	<b>IIC1 Clock Gate Control</b> 0 Bus clock to the IIC1 module is disabled. 1 Bus clock to the IIC1 module is enabled.
1 SCI2	<b>SCI2 Clock Gate Control</b> 0 Bus clock to the SCI2 module is disabled. 1 Bus clock to the SCI2 module is enabled.
1 SCI1	<b>SCI1 Clock Gate Control</b> 0 Bus clock to the SCI1 module is disabled. 1 Bus clock to the SCI1 module is enabled.

## 5.7.11 System Clock Gating Control 2 Register (SCGC2)

This register contains control bits to enable or disable the bus clock to the SCI3, FTSR, IRQ, keyboard, RTC, and SPI modules. Gating off the clocks to unused peripherals reduces the microcontroller's run and wait currents. See [Section 5.6, “Peripheral Clock Gating,”](#) for more information.

	7	6	5	4	3	2	1	0
R W	SCI3	FTSR	IRQ	KBI2	KBI1	RTC	SPI2	SPI1
Reset:	1	1	1	1	-1	1	1	1

**Figure 5-12. System Clock Gating Control 2 Register (SCGC2)**

<sup>1</sup> 1 in 80-pin and 64-pin packages, 0 in 48-pin package.

**Table 5-17. SCGC2 Register Field Descriptions**

Field	Description
7 SCI1	<b>SCI1 Clock Gate Control</b> 0 Bus clock to the SCI1 module is disabled. 1 Bus clock to the SCI1 module is enabled.
6 FTSR	<b>FTSR Clock Gate Control</b> — This bit does not affect normal program execution from the flash array. Only the clock to the flash control registers is affected. 0 Bus clock to flash registers is disabled. 1 Bus clock to flash registers is enabled.
5 IRQ	<b>IRQ Clock Gate Control</b> 0 Bus clock to the IRQ module is disabled. 1 Bus clock to the IRQ module is enabled.
4 KBI2	<b>KBI2 Clock Gate Control</b> 0 Bus clock to the KBI2 module is disabled. 1 Bus clock to the KBI2 module is enabled.
3 KBI1	<b>KBI1 Clock Gate Control</b> 0 Bus clock to the KBI1 module is disabled. 1 Bus clock to the KBI1 module is enabled.
2 RTC	<b>RTC Clock Gate Control</b> 0 Bus clock to the RTC module is disabled. 1 Bus clock to the RTC module is enabled. Only the bus clock is gated; the MGERCLK and LPOCLK are still available to the MCG
1 SPI2	<b>SPI2 Clock Gate Control</b> 0 Bus clock to the SPI2 module is disabled. 1 Bus clock to the SPI2 module is enabled.
0 SPI1	<b>SPI1 Clock Gate Control</b> 0 Bus clock to the SPI1 module is disabled. 1 Bus clock to the SPI1 module is enabled.

## 5.7.12 System Clock Gating Control 3 Register (SCGC3)

This register contains control bits to enable or disable the bus clock to the PTA-H modules. Gating off the clocks to unused peripherals reduces the microcontroller's run and wait currents. See [Section 5.6, "Peripheral Clock Gating,"](#) for more information.

	7	6	5	4	3	2	1	0
R W	PTH	PTG	PTF	PTE	PTD	PTC	PTB	PTA
Reset:	_1	_2	_2	1	1	1	1	1

**Figure 5-13. System Clock Gating Control 3 Register (SCGC3)**

- 1 1 in 80-pin package, 0 in 48-pin and 64-pin packages
- 2 1 in 80-pin and 64-pin packages, 0 in 48-pin package

**Table 5-18. SCGC3 Register Field Descriptions**

Field	Description
7 PTH	<b>PTH Clock Gate Control</b> 0 Bus clock to the PTH module is disabled. 1 Bus clock to the PTH module is enabled.
6 PTG	<b>PTG Clock Gate Control</b> 0 Bus clock to the PTG module is disabled. 1 Bus clock to the PTG module is enabled.
5 PTF	<b>PTF Clock Gate Control</b> 0 Bus clock to the PTF(RGPIO) module is disabled. 1 Bus clock to the PTF(RGPIO) module is enabled.
4 PTE	<b>PTE Clock Gate Control</b> 0 Bus clock to the PTE module is disabled. 1 Bus clock to the PTE module is enabled.
3 PTD	<b>PTD Clock Gate Control</b> 0 Bus clock to the PTD module is disabled. 1 Bus clock to the PTD module is enabled.
2 PTC	<b>PTC Clock Gate Control</b> 0 Bus clock to the PTC module is disabled. 1 Bus clock to the PTC module is enabled.
1 PTB	<b>PTB Clock Gate Control</b> 0 Bus clock to the PTB module is disabled. 1 Bus clock to the PTB module is enabled.
0 PTA	<b>PTA Clock Gate Control</b> 0 Bus clock to the PTA module is disabled. 1 Bus clock to the PTA module is enabled.

## 5.7.13 System Clock Gating Control 4 Register (SCGC4)

This register contains control bits to enable or disable the bus clock to MTIM2, Port Mux Control, the Mini-FlexBus and Fast Ethernet Controller (FEC) modules and PTJ. Gating off the clocks to unused

peripherals reduces the microcontroller's run and wait currents. See Section 5.6, “Peripheral Clock Gating,” for more information.

	7	6	5	4		3	2	1	0
R	1	1	1		MTIM2	MC	MB	FEC	PTJ
W				1		1	-1	1	-2
Reset:	1	1	1	1		1		1	2

Figure 5-14. System Clock Gating Control 3 Register (SCGC4)

<sup>1</sup> 1 in 80-pin packages, 0 in 64-pin and 48-pin package.

<sup>2</sup> 1 in 80-pin and 64-pin packages, 0 in 48-pin package.

Table 5-19. SCGC4 Register Field Descriptions

Field	Description
7-5	Reserved, must be set.
4 MTIM2	<b>MTIM2 Clock Gate Control</b> 0 Bus clock to the MTIM2 module is disabled. 1 Bus clock to the MTIM2 module is enabled.
3 MC	<b>Port Mux Control</b> 0 Bus clock to the Port Mux Control module is disabled. 1 Bus clock to the Port Mux Control module is enabled. This bit can be set to zero when the mux controls are programmed as desired. This bit affects programming of the mux controls only. The actual data paths to/from pins are unaffected.
2 MB	<b>Mini-FlexBus Clock Gate Control</b> 0 Bus clock to the Mini-FlexBus module is disabled. 1 Bus clock to the Mini-FlexBus module is enabled.
1 FEC	<b>FEC Clock Gate Control</b> 0 Bus clock to the FEC module is disabled. 1 Bus clock to the FEC module is enabled.
0 PTJ	<b>PTJ Clock Gate Control</b> 0 Bus clock to the PTJ module is disabled. 1 Bus clock to the PTJ module is enabled.

### 5.7.14 SIM Internal Peripheral Select Register (SIMIPS)

This register configures internal peripheral connections which may be routed to more than one location.

	7	6	5	4		3	2	1	0
R	0	0	0	0	TPM2	TPM1	MTIM2	MTIM1	
W					0	0	0	0	0
Reset:	0	0	0	0					0

Figure 5-15. SIM Internal Peripheral Select Register (SIMIPS)

**Table 5-20. SIMIPS Register Field Descriptions**

<b>Field</b>	<b>Description</b>
7-4	Reserved, must be cleared.
3 TPM2	<b>TPM2 External Clock Select</b> — This bit selects which external package pin supplies an external clock signal to the TPM2 module. Note that the package pin must also be configured properly using the I/O mux controls discussed in <a href="#">Section 9.7, “Pin Mux Controls.”</a> 0 TMRCLK1 1 TMRCLK2
2 TPM1	<b>TPM1 External Clock Select</b> — This bit selects which external package pin supplies an external clock signal to the TPM1 module. Note that the package pin must also be configured properly using the I/O mux controls discussed in <a href="#">Section 9.7, “Pin Mux Controls.”</a> 0 TMRCLK1 1 TMRCLK2
1 MTIM2	<b>MTIM2 External Clock Select</b> — This bit selects which external package pin supplies an external clock signal to the MTIM2 module. Note that the package pin must also be configured properly using the I/O mux controls discussed in <a href="#">Section 9.7, “Pin Mux Controls.”</a> 0 TMRCLK1 1 TMRCLK2
0 MTIM1	<b>MTIM1 External Clock Select</b> — This bit selects which external package pin supplies an external clock signal to the MTIM1 module. Note that the package pin must also be configured properly using the I/O mux controls discussed in <a href="#">Section 9.7, “Pin Mux Controls.”</a> 0 TMRCLK1 1 TMRCLK2

# **Chapter 6**

## **Multipurpose Clock Generator (MCG)**

### **6.1 Introduction**

The multipurpose clock generator (MCG) module provides several clock source choices for this device. The module contains a frequency-locked loop (FLL) and a phase-locked loop (PLL) that are controllable by either an internal or an external reference clock. The module can select either of the FLL or PLL clocks, or either of the internal or external reference clocks as a source for the MCU system clock. The selected clock source is passed through a reduced bus divider which allows a lower output clock frequency to be derived. The MCG also controls a crystal oscillator (XOSC), which allows an external crystal, ceramic resonator, or another external clock source to produce the external reference clock.

## 6.1.1 Features

Key features of the MCG module are:

- Frequency-locked loop (FLL)
  - Internal or external reference clock can be used to control the FLL
  - Phase-locked loop (PLL)
    - Voltage-controlled oscillator (VCO)
    - Modulo VCO frequency divider
    - Phase/Frequency detector
    - Integrated loop filter
    - Lock detector with interrupt capability
- Internal reference clock
  - Nine trim bits for accuracy
  - Can be selected as the clock source for the MCU
- External reference clock
  - Control for a separate crystal oscillator
  - Clock monitor with reset capability
  - Can be selected as the clock source for the MCU
- Reference divider is provided
- Clock source selected can be divided down by 1, 2, 4, or 8
- BDC clock (MCGLCLK) is provided as a constant divide-by-2 of the DCO output whether in an FLL or PLL mode. Three selectable digitally controlled oscillators (DCOs) optimized for different frequency ranges.
- Option to maximize DCO output frequency for a 32,768 Hz external reference clock source.
- The PLL can be used to drive MCGPLLSCLK even when MCGOUT is driven from one of the reference clocks (PBE mode).

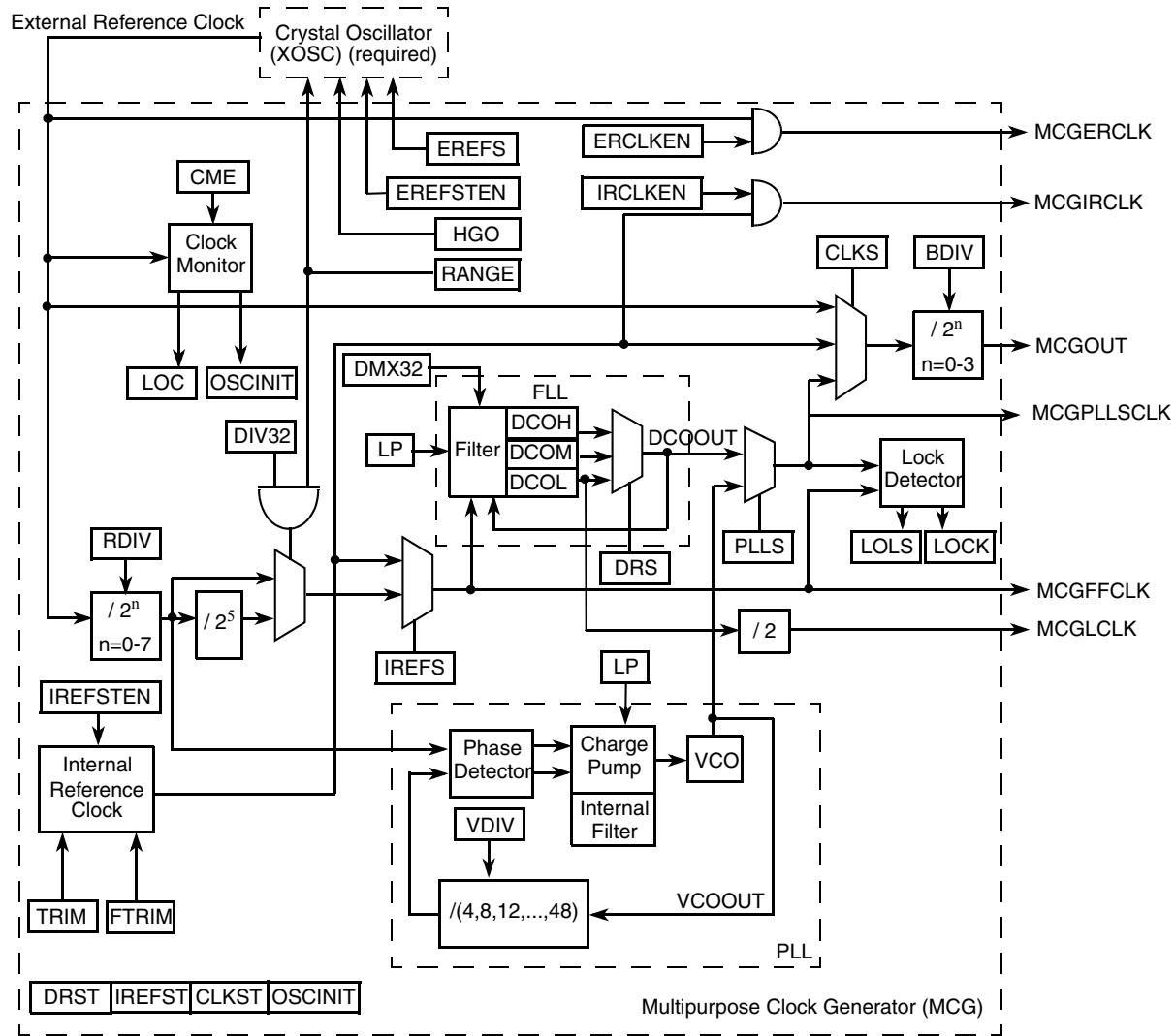


Figure 6-1. Multipurpose Clock Generator (MCG) Block Diagram

## 6.1.2 Modes of Operation

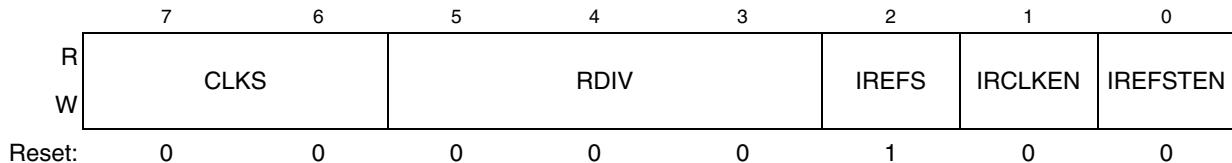
There are several modes of operation for the MCG. For details, see [Section 6.4.1, “MCG Modes of Operation.”](#)

## 6.2 External Signal Description

There are no MCG signals that connect off chip.

## 6.3 Register Definition

### 6.3.1 MCG Control Register 1 (MCGC1)



**Figure 6-2. MCG Control Register 1 (MCGC1)**

**Table 6-1. MCG Control Register 1 Field Descriptions**

Field	Description
7:6 CLKS	<b>Clock Source Select</b> — Selects the system clock source. 00 Encoding 0 — Output of FLL or PLL is selected. 01 Encoding 1 — Internal reference clock is selected. 10 Encoding 2 — External reference clock is selected. 11 Encoding 3 — Reserved, defaults to 00.
5:3 RDIV	<b>External Reference Divider</b> — Selects the amount to divide down the external reference clock. If the FLL is selected, the resulting frequency must be in the range 31.25 kHz to 39.0625 kHz. If the PLL is selected, the resulting frequency must be in the range 1 MHz to 2 MHz. See <a href="#">Table 6-2</a> and <a href="#">Table 6-3</a> for the divide-by factors.
2 IREFS	<b>Internal Reference Select</b> — Selects the reference clock source. 1 Internal reference clock selected 0 External reference clock selected
1 IRCLKEN	<b>Internal Reference Clock Enable</b> — Enables the internal reference clock for use as MCGIRCLK. 1 MCGIRCLK active 0 MCGIRCLK inactive
0 IREFSTEN	<b>Internal Reference Stop Enable</b> — Controls whether or not the internal reference clock remains enabled when the MCG enters stop mode. 1 Internal reference clock stays enabled in stop if IRCLKEN is set or if MCG is in FEI, FBI, or BLPI mode before entering stop 0 Internal reference clock is disabled in stop

**Table 6-2. FLL External Reference Divide Factor**

RDIV	Divide Factor		
	RANGE:DIV32 0:X	RANGE:DIV32 1:0	RANGE:DIV32 1:1
0	1	1	32
1	2	2	64
2	4	4	128
3	8	8	256
4	16	16	512
5	32	32	1024
6	64	64	Reserved
7	128	128	

**Table 6-3. PLL External Reference Divide Factor**

RDIV	Divide Factor
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128

### 6.3.2 MCG Control Register 2 (MCGC2)

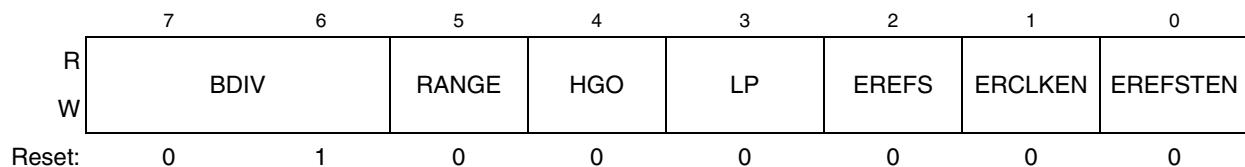
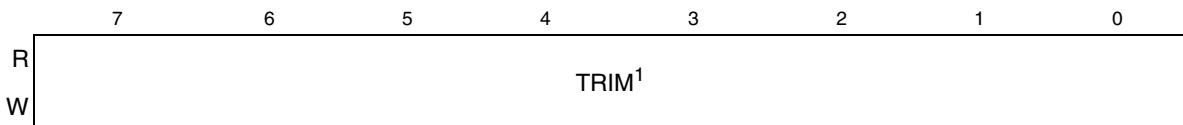


Figure 6-3. MCG Control Register 2 (MCGC2)

Table 6-4. MCG Control Register 2 Field Descriptions

Field	Description
7:6 BDIV	<b>Bus Frequency Divider</b> — Selects the amount to divide down the clock source selected by the CLKS bits in the MCGC1 register. This controls the bus frequency. 00 Encoding 0 — Divides selected clock by 1 01 Encoding 1 — Divides selected clock by 2 10 Encoding 2 — Divides selected clock by 4 11 Encoding 3 — Divides selected clock by 8
5 RANGE	<b>Frequency Range Select</b> — Selects the frequency range for the crystal oscillator or external clock source. 1 High frequency range selected for the crystal oscillator of 1 MHz to 25 MHz (1 MHz to 40 MHz for external clock source) 0 Low frequency range selected for the crystal oscillator of 32 kHz to 100 kHz (32 kHz to 1 MHz for external clock source)
4 HGO	<b>High Gain Oscillator Select</b> — Controls the crystal oscillator mode of operation. 1 Configure crystal oscillator for high gain operation 0 Configure crystal oscillator for low power operation
3 LP	<b>Low Power Select</b> — Controls whether the FLL (or PLL) is disabled in bypass modes. 1 FLL (or PLL) is disabled in bypass modes (lower power). 0 FLL (or PLL) is not disabled in bypass modes.
2 EREFNS	<b>External Reference Select</b> — Selects the source for the external reference clock. 1 Oscillator requested 0 External Clock Source requested
1 ERCLKEN	<b>External Reference Enable</b> — Enables the external reference clock for use as MCGERCLK. 1 MCGERCLK active 0 MCGERCLK inactive
0 EREFSTEN	<b>External Reference Stop Enable</b> — Controls whether or not the external reference clock remains enabled when the MCG enters stop mode. 1 External reference clock stays enabled in stop if ERCLKEN is set or if MCG is in FEE, FBE, PEE, PBE, or BLPE mode before entering stop 0 External reference clock is disabled in stop

### 6.3.3 MCG Trim Register (MCGTRM)



**Figure 6-4. MCG Trim Register (MCGTRM)**

<sup>1</sup> A value for TRIM is loaded during reset from a factory programmed location when not in any BDM mode. If in a BDM mode, a default value of 0x80 is loaded.

**Table 6-5. MCG Trim Register Field Descriptions**

Field	Description
7:0 TRIM	<p><b>MCG Trim Setting</b> — Controls the internal reference clock frequency by controlling the internal reference clock period. The TRIM bits are binary weighted (i.e., bit 1 adjusts twice as much as bit 0). Increasing the binary value in TRIM increases the period, and decreasing the value decreases the period.</p> <p>An additional fine trim bit is available in MCGSC as the FTRIM bit.</p> <p>If a TRIM[7:0] value stored in nonvolatile memory is to be used, it's the user's responsibility to copy that value from the nonvolatile memory location to this register.</p>

### 6.3.4 MCG Status and Control Register (MCGSC)

	7	6	5	4	3	2	1	0
R	LOLS	LOCK	PLLST	IREFST	CLKST	OSCINIT		FTRIM <sup>1</sup>
W							0	
Reset:	0	0	0	1	0	0	0	

Figure 6-5. MCG Status and Control Register (MCGSC)

<sup>1</sup> A value for FTRIM is loaded during reset from a factory programmed location when not in any BDM mode. If in a BDM mode, a default value of 1'b0 is loaded.

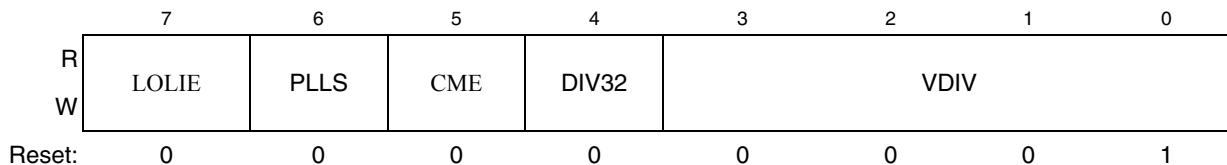
Table 6-6. MCG Status and Control Register Field Descriptions

Field	Description
7 LOLS	<b>Loss of Lock Status</b> — This bit is a sticky indication of lock status for the FLL or PLL. LOLS is set when lock detection is enabled and after acquiring lock, the FLL or PLL output frequency has fallen outside the lock exit frequency tolerance, $D_{unl}$ . LOLIE determines whether an interrupt request is made when set. LOLS is cleared by reset or by writing a logic 1 to LOLS when LOLS is set. Writing a logic 0 to LOLS has no effect. 0 FLL or PLL has not lost lock since LOLS was last cleared. 1 FLL or PLL has lost lock since LOLS was last cleared.
6 LOCK	<b>Lock Status</b> — Indicates whether the FLL or PLL has acquired lock. Lock detection is disabled when the FLL and PLL are disabled. If the lock status bit is set, changing the value of DMX32, DRS[1:0] and IREFS bits in FBE, FBI, FEE and FEI modes; DIV32 bit in FBE and FEE modes; TRIM[7:0] bits in FBI and FEI modes; RDIV[2:0] bits in FBE, FEE, PBE and PEE modes; VDIV[3:0] bits in PBE and PEE modes; and PLLS bit, causes the lock status bit to clear and stay clear until the FLL or PLL has reacquired lock. Entry into BLPI, BLPE or stop mode also causes the lock status bit to clear and stay cleared until the exit of these modes and the FLL or PLL has reacquired lock. 0 FLL or PLL is currently unlocked. 1 FLL or PLL is currently locked.
5 PLLST	<b>PLL Select Status</b> — The PLLST bit indicates the current source for the PLLS clock. The PLLST bit does not update immediately after a write to the PLLS bit due to internal synchronization between clock domains. 0 Source of PLLS clock is FLL clock. 1 Source of PLLS clock is PLL clock.
4 IREFST	<b>Internal Reference Status</b> — The IREFST bit indicates the current source for the reference clock. The IREFST bit does not update immediately after a write to the IREFS bit due to internal synchronization between clock domains. 0 Source of reference clock is external reference clock (oscillator or external clock source as determined by the EREFS bit in the MCGC2 register). 1 Source of reference clock is internal reference clock.
3:2 CLKST	<b>Clock Mode Status</b> — The CLKST bits indicate the current clock mode. The CLKST bits do not update immediately after a write to the CLKS bits due to internal synchronization between clock domains. 00 Encoding 0 — Output of FLL is selected. 01 Encoding 1 — Internal reference clock is selected. 10 Encoding 2 — External reference clock is selected. 11 Encoding 3 — Output of PLL is selected.

**Table 6-6. MCG Status and Control Register Field Descriptions (continued)**

Field	Description
1 OSCINIT	<b>OSC Initialization</b> — If the external reference clock is selected by ERCLKEN or by the MCG being in FEE, FBE, PEE, PBE, or BLPE mode, and if EREFS is set, then this bit is set after the initialization cycles of the crystal oscillator clock have completed. This bit is only cleared when EREFS is cleared or when the MCG is in FEI, FBI, or BLPI mode and ERCLKEN is cleared.
0 FTRIM	<b>MCG Fine Trim</b> — Controls the smallest adjustment of the internal reference clock frequency. Setting FTRIM increases the period and clearing FTRIM decreases the period by the smallest amount possible.  If an FTRIM value stored in nonvolatile memory is to be used, it's the user's responsibility to copy that value from the nonvolatile memory location to this register's FTRIM bit.

### 6.3.5 MCG Control Register 3 (MCGC3)

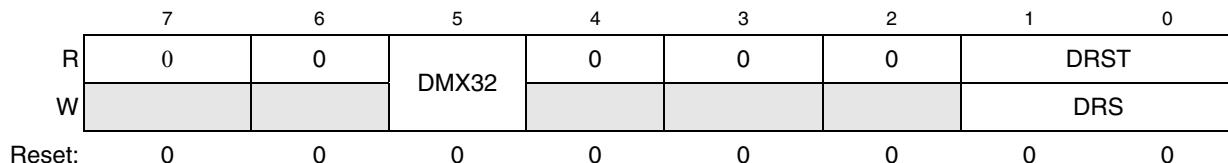
**Figure 6-6. MCG PLL Register (MCGPLL)****Table 6-7. MCG PLL Register Field Descriptions**

Field	Description
7 LOLIE	<b>Loss of Lock Interrupt Enable</b> — Determines if an interrupt request is made following a loss of lock indication. The LOLIE bit only has an effect when LOLS is set. 0 No request on loss of lock. 1 Generate an interrupt request on loss of lock.
6 PLLS	<b>PLL Select</b> — Controls whether the PLL or FLL is selected. If the PLLS bit is clear, the PLL is disabled in all modes. If the PLLS is set, the FLL is disabled in all modes. This bit should only be written in bypassed external modes when CLKST=10. 1 PLL is selected 0 FLL is selected
5 CME	<b>Clock Monitor Enable</b> — Determines if a reset request is made following a loss of external clock indication. The CME bit should only be set to a logic 1 when the MCG is in an operational mode that uses the external clock (FEE, FBE, PEE, PBE, or BLPE) or the external reference is enabled (ERCLKEN=1 in the MCGC2 register). When the CME bit is set to a logic 1, the value of the RANGE bit in the MCGC2 register should not be changed. If the external reference clock is set to be disabled when the MCG enters STOP mode (EREFSTEN=0), then the CME bit should be set to a logic 0 before the MCG enters STOP mode. Otherwise, a reset request may occur while in STOP mode. 0 Clock monitor is disabled. 1 Generate a reset request on loss of external clock.

**Table 6-7. MCG PLL Register Field Descriptions (continued)**

Field	Description
4 DIV32	<b>Divide-by-32 Enable</b> — Controls an additional divide-by-32 factor to the external reference clock for the FLL when RANGE bit is set. When the RANGE bit is 0, this bit has no effect. Writes to this bit are ignored if PLLS bit is set. 0 Divide-by-32 is disabled. 1 Divide-by-32 is enabled when RANGE=1.
3:0 VDIV	<b>VCO Divider</b> — Selects the amount to divide down the VCO output of PLL. The VDIV bits establish the multiplication factor (M) applied to the reference clock frequency. 0000 Encoding 0 — Reserved. 0001 Encoding 1 — Multiply by 4. 0010 Encoding 2 — Multiply by 8. 0011 Encoding 3 — Multiply by 12. 0100 Encoding 4 — Multiply by 16. 0101 Encoding 5 — Multiply by 20. 0110 Encoding 6 — Multiply by 24. 0111 Encoding 7 — Multiply by 28. 1000 Encoding 8 — Multiply by 32. 1001 Encoding 9 — Multiply by 36. 1010 Encoding 10 — Multiply by 40. 1011 Encoding 11 — Multiply by 44. 1100 Encoding 12 — Multiply by 48. 1101 Encoding 13 — Reserved (default to M=48). 111x Encoding 14-15 — Reserved (default to M=48).

### 6.3.6 MCG Control Register 4 (MCGC4)

**Figure 6-7. MCG Control Register 4 (MCGC4)**

**Table 6-8. MCG Test and Control Register Field Descriptions**

Field	Description
7:6	Reserved for test, user code should not write 1's to these bits.
5 DMX32	<b>DCO Maximum frequency with 32.768 kHz reference</b> — The DMX32 bit controls whether or not the DCO frequency range is narrowed to its maximum frequency with a 32.768 kHz reference. See <a href="#">Table 6-9</a> . 0 DCO has default range of 25%. 1 DCO is fined tuned for maximum frequency with 32.768 kHz reference.
4:2	Reserved for test, user code should not write 1's to these bits.
1:0 DRST DRS	<b>DCO Range Status</b> — The DRST read bits indicate the current frequency range for the FLL output, DCOOUT. See <a href="#">Table 6-9</a> . The DRST bits do not update immediately after a write to the DRS field due to internal synchronization between clock domains. The DRST bits are not valid in BLPI, BLPE, PBE or PEE mode and it reads zero regardless of the DCO range selected by the DRS bits.  <b>DCO Range Select</b> — The DRS bits select the frequency range for the FLL output, DCOOUT. Writes to the DRS bits while the LP or PLLS bit is set are ignored. 00 Low range. 01 Mid range. 10 High range. 11 Reserved

**Table 6-9. DCO frequency range<sup>1</sup>**

DRS	DMX32	Reference range	FLL factor	DCO range
00	0	31.25 - 39.0625 kHz	512	16 - 20 MHz
	1	32.768 kHz	608	19.92 MHz
01	0	31.25 - 39.0625 kHz	1024	32 - 40 MHz
	1	32.768 kHz	1216	39.85 MHz
10	0	31.25 - 39.0625 kHz	1536	48-60 MHz <sup>1</sup>
	0	32.768 kHz		50.33 MHz <sup>1</sup>
	1	32.768 kHz	1824	59.77 MHz <sup>1</sup>
11		Reserved		

<sup>1</sup> The resulting bus clock frequency should not exceed the maximum specified bus clock frequency of the device.

## 6.4 Functional Description

### 6.4.1 MCG Modes of Operation

The MCG operates in one of the modes described in [Table 6-10](#).

#### NOTE

The MCG restricts transitions between modes. For the permitted transitions, see [Section 6.4.2, “MCG Mode State Diagram.”](#)

**Table 6-10. MCG Modes of Operation**

<b>Mode</b>	<b>Related field values</b>	<b>Description</b>
FLL Engaged Internal (FEI)	<ul style="list-style-type: none"> <li>MCGC1[IREFS] = 1</li> <li>MCGC1[CLKS] = 00</li> <li>MCGC3[PLLS] = 0</li> </ul>	Default. MCGOUT is derived from the FLL clock, which is controlled by the internal reference clock. The FLL clock frequency locks to 1024 times the internal reference frequency. MCGLCLK is derived from the FLL, and the PLL is disabled in a low-power state.
FLL Engaged External (FEE)	<ul style="list-style-type: none"> <li>MCGC1[IREFS] = 0</li> <li>MCGC1[CLKS] = 00</li> <li>MCGC1[RDIV] is programmed to divide the reference clock to be within the range of 31.2500 to 39.0625 kHz.</li> <li>MCGC3[PLLS] = 0</li> </ul>	MCGOUT is derived from the FLL clock, which is controlled by the external reference clock. The external reference clock that is enabled can be produced by an external crystal, ceramic resonator, or another external clock source connected to the required crystal oscillator (XOSC). The FLL clock frequency locks to 1024 times the external reference frequency, as specified by MCGC1[RDIV], MCGC2[RANGE], and MCGC3[DIV32]. MCGLCLK is derived from the FLL, and the PLL is disabled in a low-power state.
FLL Bypassed Internal (FBI)	<ul style="list-style-type: none"> <li>MCGC1[IREFS] = 1</li> <li>MCGC1[CLKS] = 01</li> <li>MCGC2[LP] = 0 (or the BDM is enabled)</li> <li>MCGC3[PLLS] = 0</li> </ul>	<p>MCGOUT is derived from the internal reference clock; the FLL is operational, but its output clock is not used. This mode is useful to allow the FLL to acquire its target frequency while the MCGOUT clock is driven from the internal reference clock.</p> <p>MCGOUT is derived from the internal reference clock. The FLL clock is controlled by the internal reference clock, and the FLL clock frequency locks to 1024 times the internal reference frequency. MCGLCLK is derived from the FLL, and the PLL is disabled in a low-power state.</p>
FLL Bypassed External (FBE)	<ul style="list-style-type: none"> <li>MCGC1[IREFS] = 0</li> <li>MCGC1[CLKS] = 10</li> <li>MCGC1[RDIV] is programmed to divide the reference clock to be within the range of 31.2500 to 39.0625 kHz</li> <li>MCGC2[LP] = 0 (or the BDM is enabled)</li> <li>MCGC3[PLLS] = 0</li> </ul>	<p>MCGOUT is derived from the external reference clock; the FLL is operational, but its output clock is not used. This mode is useful to allow the FLL to acquire its target frequency while MCGOUT is driven from the external reference clock.</p> <p>MCGOUT is derived from the external reference clock. The external reference clock that is enabled can be produced by an external crystal, ceramic resonator, or another external clock source connected to the required crystal oscillator (XOSC). The FLL clock is controlled by the external reference clock, and the FLL clock frequency locks to 1024 times the external reference frequency, as selected by MCGC1[RDIV], MCGC2[RANGE], and MCGC3[DIV32]. MCGLCLK is derived from the FLL, and the PLL is disabled in a low-power state.</p>
PLL Engaged External (PEE)	<ul style="list-style-type: none"> <li>MCGC1[IREFS] = 0</li> <li>MCGC1[CLKS] = 00</li> <li>MCGC1[RDIV] is programmed to divide the reference clock to be within the range of 1 to 2 MHz.</li> <li>PLLS = 1</li> </ul>	MCGOUT is derived from the PLL clock, which is controlled by the external reference clock. The external reference clock that is enabled can be produced by an external crystal, ceramic resonator, or another external clock source connected to the required crystal oscillator (XOSC). The PLL clock frequency locks to a multiplication factor, as specified by MCGC3[VDIV], times the external reference frequency, as specified by MCGC1[RDIV], MCGC2[RANGE], and MCGC3[DIV32]. If the BDM is enabled, MCGLCLK is derived from the DCO (open-loop mode) divided by two. If the BDM is not enabled, the FLL is disabled in a low-power state.

**Table 6-10. MCG Modes of Operation (continued)**

Mode	Related field values	Description
PLL Bypassed External (PBE)	<ul style="list-style-type: none"> <li>MCGC1[IREFS] = 0</li> <li>MCGC1[CLKS] = 10</li> <li>MCGC1[RDIV] is programmed to divide the reference clock to be within the range of 1 to 2 MHz.</li> <li>MCGC2[LP] = 0</li> <li>MCGC3[PLLS] = 1</li> </ul>	<p>MCGOUT is derived from the external reference clock; the PLL is operational, but its output clock is not used. This mode is useful to allow the PLL to acquire its target frequency while MCGOUT is driven from the external reference clock.</p> <p>MCGOUT is derived from the external reference clock. The external reference clock that is enabled can be produced by an external crystal, ceramic resonator, or another external clock source connected to the required crystal oscillator (XOSC). The PLL clock frequency locks to a multiplication factor, as specified by MCGC3[VDIV], times the external reference frequency, as specified by MCGC1[RDIV], MCGC2[RANGE], and MCGC3[DIV32]. If the BDM is enabled, MCGLCLK is derived from the DCO (open-loop mode) divided by two. If the BDM is not enabled, the FLL is disabled in a low-power state.</p>
Bypassed Low Power Internal (BLPI)	<ul style="list-style-type: none"> <li>MCGC1[IREFS] = 1</li> <li>MCGC1[CLKS] = 01</li> <li>MCGC2[LP] = 1 (and the BDM is disabled)</li> </ul>	<p>MCGOUT is derived from the internal reference clock. The PLL and FLL are disabled, and MCGLCLK is not available for BDC communications. If the BDM becomes enabled, the mode switches to one of the bypassed internal modes as determined by the state of MCGC3[PLLS].</p>
Bypassed Low Power External (BLPE)	<ul style="list-style-type: none"> <li>MCGC1[IREFS] = 0</li> <li>MCGC1[CLKS] = 10</li> <li>MCGC2[LP] = 1 (and the BDM is disabled)</li> </ul>	<p>MCGOUT is derived from the external reference clock. The external reference clock that is enabled can be produced by an external crystal, ceramic resonator, or another external clock source connected to the required crystal oscillator (XOSC).</p> <p>The PLL and FLL are disabled, and MCGLCLK is not available for BDC communications. If the BDM becomes enabled, the mode switches to one of the bypassed external modes as determined by the state of MCGC3[PLLS].</p>
Stop	—	<p>Entered when the MCU enters a Stop state. The FLL and PLL are disabled, and all MCG clock signals are static except in the following cases:</p> <p>MCGIRCLK is active in Stop mode when all the following conditions become true:</p> <ul style="list-style-type: none"> <li>MCGC1[IRCLKEN] = 1</li> <li>MCGC1[IREFSTEN] = 1</li> </ul> <p>MGERCLK is active in Stop mode when all the following conditions become true:</p> <ul style="list-style-type: none"> <li>MCGC2[ERCLKEN] = 1</li> <li>MCGC2[EREFTEN] = 1</li> </ul>

## 6.4.2 MCG Mode State Diagram

Figure 6-8 shows the MCG's mode state diagram. The arrows indicate the permitted mode transitions.

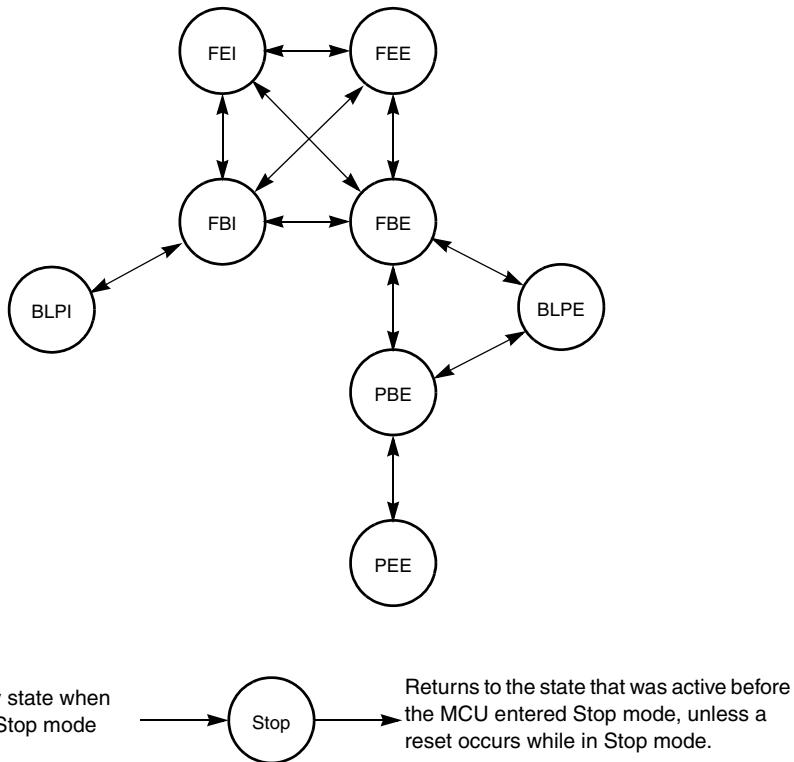


Figure 6-8. MCG Mode State Diagram

### 6.4.3 Mode Switching

The IREFS bit can be changed at anytime, but the actual switch to the newly selected clock is shown by the IREFST bit. When switching between engaged internal and engaged external modes, the FLL or PLL begins locking again after the switch is completed.

The CLKS bits can also be changed at anytime, but the actual switch to the newly selected clock is shown by the CLKST bits. If the newly selected clock is not available, the previous clock remains selected.

The DRS bits can be changed at anytime except when LP bit is 1. If the DRS bits are changed while in FLL engaged internal (FEI) or FLL engaged external (FEE), the bus clock remains at the previous DCO range until the new DCO starts. When the new DCO starts the bus clock switches to it. After switching to the new DCO the FLL remains unlocked for several reference cycles. After the selected DCO startup time is over, the FLL is locked. The completion of the switch is shown by the DRST bits.

For details, see [Figure 6-8](#).

#### 6.4.4 Bus Frequency Divider

The BDIV bits can be changed at anytime and the actual switch to the new frequency occurs immediately.

#### 6.4.5 Low Power Bit Usage

The low power bit (LP) is provided to allow the FLL or PLL to be disabled and thus conserve power when these systems are not being used. The DRS bit can not be written while LP bit is 1. However, in some applications it may be desirable to enable the FLL or PLL and allow it to lock for maximum accuracy before switching to an engaged mode. Do this by writing the LP bit to 0.

#### 6.4.6 Internal Reference Clock

When IRCLKEN is set, the internal reference clock signal is presented as MCGIRCLK, which can be used as an additional clock source. The MCGIRCLK frequency can be re-targeted by trimming the period of the internal reference clock. This can be done by writing a new value to the TRIM bits in the MCGTRM register. Writing a larger value decreases the MCGIRCLK frequency, and writing a smaller value to the MCGTRM register increases the MCGIRCLK frequency. The TRIM bits affects the MCGOUT frequency if the MCG is in FLL engaged internal (FEI), FLL bypassed internal (FBI), or bypassed low power internal (BLPI) mode. The TRIM and FTRIM value is initialized by POR but is not affected by other resets.

If IREFSTEN and IRCLKEN bits are set, the internal reference clock keeps running during stop mode to provide a fast recovery upon exiting stop.

All MCU devices are factory programmed with a trim value in a reserved memory location. This value can be copied to the MCGTRM register during reset initialization. The factory trim value does not include the FTRIM bit. For finer precision, the user can trim the internal oscillator in the application and set the FTRIM bit accordingly.

#### 6.4.7 External Reference Clock

The MCG module can support an external reference clock with frequencies between 31.25 kHz to 40 MHz in all modes. When ERCLKEN is set, the external reference clock signal is presented as MGERCLK, which can be used as an additional clock source. When IREFS = 1, the external reference clock is not used by the FLL or PLL and is only used as MGERCLK. In these modes, the frequency can be equal to the maximum frequency the chip-level timing specifications supports (see [Chapter 1, “Device Overview”](#)).

If EREFSTEN and ERCLKEN bits are set or the MCG is in FEE, FBE, PEE, PBE or BLPE mode, the external reference clock keeps running during stop mode to provide a fast recovery upon exiting stop.

If CME bit is written to 1, the clock monitor is enabled. If the external reference falls below a certain frequency ( $f_{loc\_high}$  or  $f_{loc\_low}$  depending on the RANGE bit in the MCGC2), the MCU resets. The LOC bit in the System Reset Status (SRS) register is set to indicate the error.

### 6.4.8 Fixed Frequency Clock

The MCG presents the divided reference clock as MCGFFCLK for use as an additional clock source. The MCGFFCLK frequency must be no more than 1/4 of the MCGOUT frequency to be valid. This clock is intended for use in systems that include a USB interface. It allows the MCG to supply a 48MHz clock to the USB. This same clock can be used to derive MCGOUT. Alternately, MCGOUT can be derived from internal or external reference clock. This allows the CPU to run at a lower frequency (to conserve power) while the USB continues to monitor traffic.

The FLL can not be used for generation of the system clocks while the PLL is supplying MCGPLLSCLK.

## 6.5 Initialization / Application Information

This section describes how to initialize and configure the MCG module in application. The following sections include examples on how to initialize the MCG and properly switch between the various available modes.

### 6.5.1 MCG Module Initialization Sequence

The MCG comes out of reset configured for FEI mode with the BDIV set for divide-by-2. The internal reference stabilizes in  $t_{irefst}$  microseconds before the FLL can acquire lock. As soon as the internal reference is stable, the FLL acquires lock in  $t_{fll\_acquire}$  milliseconds.

#### NOTE

If the internal reference is not already trimmed, the BDIV value should not be changed to divide-by-1 without first trimming the internal reference. Failure to do so could result in the MCU running out of specification.

#### 6.5.1.1 Initializing the MCG

Because the MCG comes out of reset in FEI mode, the only MCG modes that can be directly switched to upon reset are FEE, FBE, and FBI modes (see [Figure 6-8](#)). Reaching any of the other modes requires first configuring the MCG for one of these three initial modes. Care must be taken to check relevant status bits in the MCGSC register reflecting all configuration changes within each mode.

To change from FEI mode to FEE or FBE modes, follow this procedure:

1. Enable the external clock source by setting the appropriate bits in MCGC2.
2. If the RANGE bit (bit 5) in MCGC2 is set, set DIV32 in MCGC3 to allow access to the proper RDIV values.
3. Write to MCGC1 to select the clock mode.
  - If entering FEE mode, set RDIV appropriately, clear the IREFS bit to switch to the external reference, and leave the CLKS bits at %00 so that the output of the FLL is selected as the system clock source.
  - If entering FBE, clear the IREFS bit to switch to the external reference and change the CLKS bits to %10 so that the external reference clock is selected as the system clock source. The

RDIV bits should also be set appropriately here according to the external reference frequency because although the FLL is bypassed, it remains on in FBE mode.

- The internal reference can optionally be kept running by setting the IRCLKEN bit. This is useful if the application switches back and forth between internal and external modes. For minimum power consumption, leave the internal reference disabled while in an external clock mode.
4. After the proper configuration bits have been set, wait for the affected bits in the MCGSC register to be changed appropriately, reflecting that the MCG has moved into the proper mode.
    - If ERCLKEN was set in step 1 or the MCG is in FEE, FBE, PEE, PBE, or BLPE mode, and EREFS was also set in step 1, wait here for the OSCINIT bit to become set indicating that the external clock source has finished its initialization cycles and stabilized.
    - If in FEE mode, check to make sure the IREFST bit is cleared and the LOCK bit is set before moving on.
    - If in FBE mode, check to make sure the IREFST bit is cleared, the LOCK bit is set, and the CLKST bits have changed to %10 indicating the external reference clock has been appropriately selected. Although the FLL is bypassed in FBE mode, it remains on and locks in FBE mode.
  5. Write to the MCGC4 register to determine the DCO output (MCGOUT) frequency range. Make sure that the resulting bus clock frequency does not exceed the maximum specified bus clock frequency of the device.
    - By default, with DMX32 cleared to 0, the FLL multiplier for the DCO output is 512. For greater flexibility, if a mid-range FLL multiplier of 1024 is desired instead, set the DRS[1:0] bits to %01 for a DCO output frequency of 33.55 MHz. If a high-range FLL multiplier of 1536 is desired instead, set the DRS[1:0] bits to %10 for a DCO output frequency of 50.33 MHz.
    - When using a 32.768 kHz external reference, if the maximum low-range DCO frequency that can be achieved with a 32.768 kHz reference is desired, set the DRS[1:0] bits to %00 and set the DMX32 bit to 1. The resulting DCO output (MCGOUT) frequency with the new multiplier of 608 is 19.92 MHz.
    - When using a 32.768 kHz external reference, if the maximum mid-range DCO frequency that can be achieved with a 32.768 kHz reference is desired, set the DRS[1:0] bits to %01 and set the DMX32 bit to 1. The resulting DCO output (MCGOUT) frequency with the new multiplier of 1216 is 39.85 MHz.
    - When using a 32.768 kHz external reference, if the maximum high-range DCO frequency that can be achieved with a 32.768 kHz reference is desired, set the DRS[1:0] bits to %10 and set the DMX32 bit to 1. The resulting DCO output (MCGOUT) frequency with the new multiplier of 1824 is 59.77 MHz.
  6. Wait for the LOCK bit in MCGSC to become set, indicating that the FLL has locked to the new multiplier value designated by the DRS and DMX32 bits.

#### NOTE

Setting DIV32 (bit 4) in MCGC3 is strongly recommended for FLL external modes when using a high frequency range (RANGE = 1) external reference clock. The DIV32 bit is ignored in all other modes.

To change from FEI clock mode to FBI clock mode, follow this procedure:

1. Change the CLKS bits in MCGC1 to %01 so that the internal reference clock is selected as the system clock source.
2. Wait for the CLKST bits in the MCGSC register to change to %01, indicating that the internal reference clock has been appropriately selected.

### 6.5.2 Using a 32.768 kHz Reference

In FEE and FBE modes, if using a 32.768 kHz external reference, at the default FLL multiplication factor of 512, the DCO output (MCGOUT) frequency is 16.78 MHz at high-range. If the DRS[1:0] bits are set to %01, the multiplication factor is doubled to 1024, and the resulting DCO output frequency is 33.55 MHz at mid-range. If the DRS[1:0] bits are set to %10, the multiplication factor is set to 1536, and the resulting DCO output frequency is 50.33 MHz at high-range. Make sure that the resulting bus clock frequency does not exceed the maximum specified bus clock frequency of the device.

Setting the DMX32 bit in MCGC4 to 1 increases the FLL multiplication factor to allow the 32.768 kHz reference to achieve its maximum DCO output frequency. When the DRS[1:0] bits are set to %00, the 32.768 kHz reference can achieve a high-range maximum DCO output of 19.92 MHz with a multiplier of 608. When the DRS[1:0] bits are set to %01, the 32.768 kHz reference can achieve a mid-range maximum DCO output of 39.85 MHz with a multiplier of 1216. When the DRS[1:0] bits are set to %10, the 32.768 kHz reference can achieve a high-range maximum DCO output of 59.77 MHz with a multiplier of 1824. Make sure that the resulting bus clock frequency does not exceed the maximum specified bus clock frequency of the device.

In FBI and FEI modes, setting the DMX32 bit is not recommended. If the internal reference is trimmed to a frequency above 32.768 kHz, the greater FLL multiplication factor could potentially push the microcontroller system clock out of specification and damage the part.

### 6.5.3 MCG Mode Switching

When switching between operational modes of the MCG, certain configuration bits must be changed to properly move from one mode to another. Each time any of these bits are changed (PLLS, IREFS, CLKS, or EREFS), the corresponding bits in the MCGSC register (PLLST, IREFST, CLKST, or OSCINIT) must be checked before moving on in the application software.

Additionally, care must be taken to ensure that the reference clock divider (RDIV) is set properly for the mode being switched to. For instance, in PEE mode, if using a 4 MHz crystal, RDIV must be set to %001 (divide-by-2) or %010 (divide -by-4) to divide the external reference down to the required frequency between 1 and 2 MHz.

If switching to FBE or FEE mode, first setting the DIV32 bit ensures a proper reference frequency is sent to the FLL clock at all times.

In FBE, FEE, FBI, and FEI modes, at any time, the application can switch the FLL multiplication factor between 512, 1024, and 1536 with the DRS[1:0] bits in MCGC4. Writes to the DRS[1:0] bits is ignored if LP=1 or PLLS=1.

The RDIV and IREFS bits should always be set properly before changing the PLLS bit so that the FLL or PLL clock has an appropriate reference clock frequency to switch to. The table below shows MCGOUT frequency calculations using RDIV, BDIV, and VDIV settings for each clock mode. The bus frequency is equal to MCGOUT divided by 2.

**Table 6-11. MCGOUT Frequency Calculation Options**

Clock Mode	$f_{MCGOUT}^1$	Note
FEI (FLL engaged internal)	$(f_{int} * F) / B$	Typical $f_{MCGOUT} = 16$ MHz immediately after reset.
FEE (FLL engaged external)	$(f_{ext} / R * F) / B$	$f_{ext} / R$ must be in the range of 31.25 kHz to 39.0625 kHz
FBE (FLL bypassed external)	$f_{ext} / B$	$f_{ext} / R$ must be in the range of 31.25 kHz to 39.0625 kHz
FBI (FLL bypassed internal)	$f_{int} / B$	Typical $f_{int} = 32$ kHz
PEE (PLL engaged external)	$[(f_{ext} / R) * M] / B$	$f_{ext} / R$ must be in the range of 1 MHz to 2 MHz
PBE (PLL bypassed external)	$f_{ext} / B$	$f_{ext} / R$ must be in the range of 1 MHz to 2 MHz
BLPI (Bypassed low power internal)	$f_{int} / B$	
BLPE (Bypassed low power external)	$f_{ext} / B$	

<sup>1</sup>R is the reference divider selected by the RDIV bits, B is the bus frequency divider selected by the BDIV bits, F is the FLL factor selected by the DRS[1:0] and DMX32 bits, and M is the multiplier selected by the VDIV bits.

This section includes three mode switching examples using an 8 MHz external crystal. If using an external clock source less than 1 MHz, the MCG should not be configured for any of the PLL modes (PEE and PBE).

### 6.5.3.1 Example 1: Moving from FEI to PEE Mode: External Crystal = 8 MHz, Bus Frequency = 16 MHz

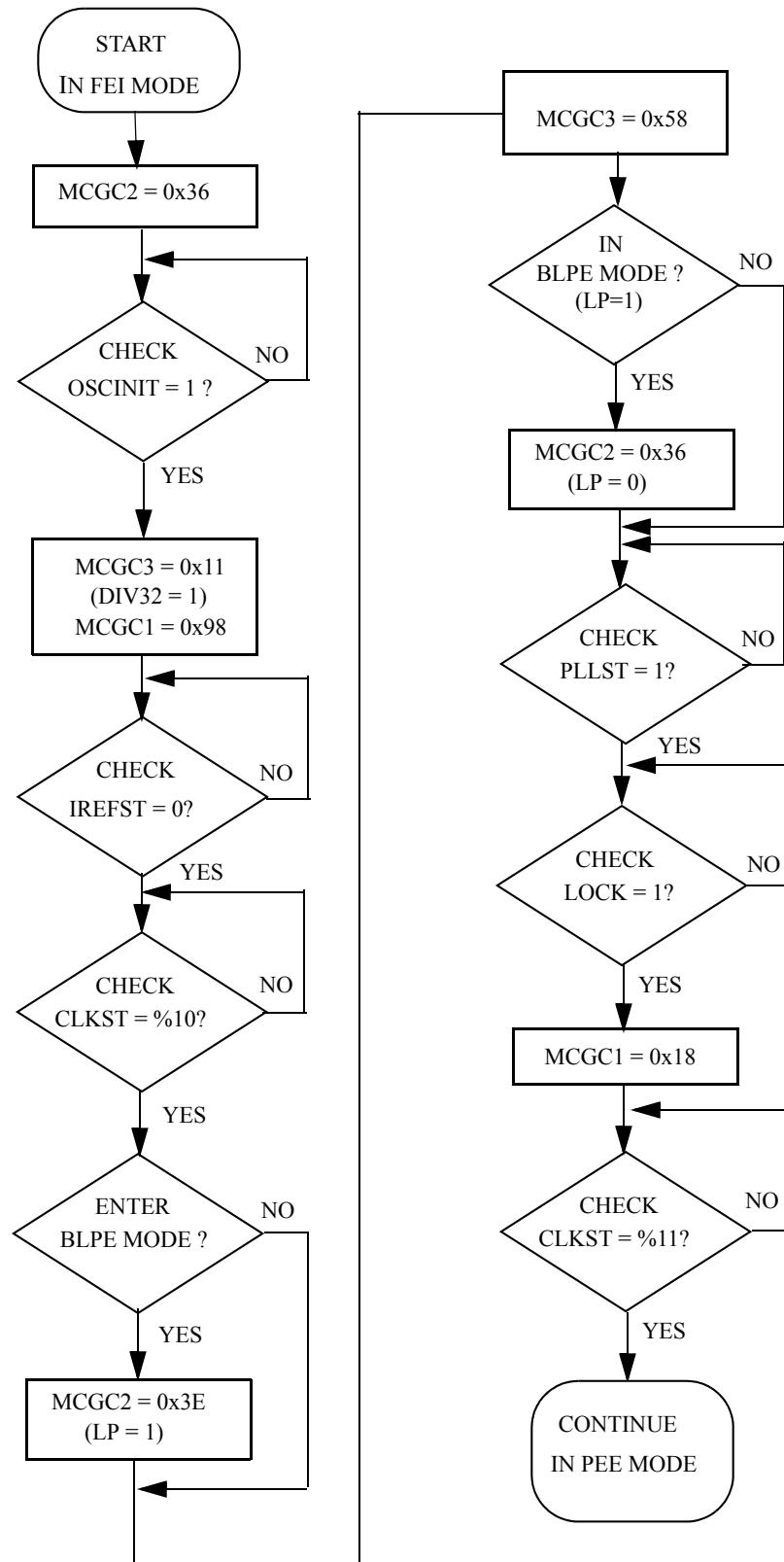
In this example, the MCG moves through the proper operational modes from FEI to PEE mode until the 8 MHz crystal reference frequency is set to achieve a bus frequency of 16 MHz. Because the MCG is in FEI mode out of reset, this example also shows how to initialize the MCG for PEE mode out of reset. First, the code sequence is described. Then a flowchart is included that illustrates the sequence.

1. First, FEI must transition to FBE mode:
  - a)  $MCGC2 = 0x36$  (%00110110)
    - BDIV (bits 7 and 6) set to %00, or divide-by-1
    - RANGE (bit 5) set to 1 because the frequency of 8 MHz is within the high frequency range
    - HGO (bit 4) set to 1 to configure the crystal oscillator for high gain operation
    - EREFS (bit 2) set to 1, because a crystal is being used
    - ERCLKEN (bit 1) set to 1 to ensure the external reference clock is active

- b) Loop until OSCINIT (bit 1) in MCGSC is 1, indicating the crystal selected by the EREFS bit has been initialized.
  - c) Because RANGE = 1, set DIV32 (bit 4) in MCGC3 to allow access to the proper RDIV bits while in an FLL external mode.
  - d) MCGC1 = 0x98 (%10011000)
    - CLKS (bits 7 and 6) set to %10 to select external reference clock as system clock source
    - RDIV (bits 5-3) set to %011, or divide-by-256 because  $8\text{MHz} / 256 = 31.25\text{ kHz}$  that is in the 31.25 kHz to 39.0625 kHz range required by the FLL
    - IREFS (bit 2) cleared to 0, selecting the external reference clock
  - e) Loop until IREFST (bit 4) in MCGSC is 0, indicating the external reference is the current source for the reference clock
  - f) Loop until CLKST (bits 3 and 2) in MCGSC is %10, indicating that the external reference clock is selected to feed MCGOUT
2. Then, FBE must transition directly to PBE mode or first through BLPE mode and then to PBE mode:
    - a) BLPE: If a transition through BLPE mode is desired, first set LP (bit 3) in MCGC2 to 1.
    - b) BLPE/PBE: MCGC3 = 0x58 (%01011000)
      - PLLS (bit 6) set to 1, selects the PLL. At this time, with an RDIV value of %011, the FLL reference divider of 256 is switched to the PLL reference divider of 8 (see [Table 6-3](#)), resulting in a reference frequency of  $8\text{ MHz} / 8 = 1\text{ MHz}$ . In BLPE mode, changing the PLLS bit only prepares the MCG for PLL usage in PBE mode
      - DIV32 (bit 4) remains set at 1. Because the MCG is in a PLL mode, the DIV32 bit is ignored. Keeping it set at 1 makes transitions back into an FLL external mode easier.
      - VDIV (bits 3-0) set to %1000, or multiply-by-32 because  $1\text{ MHz}$  reference \* 32 = 32MHz. In BLPE mode, the configuration of the VDIV bits does not matter because the PLL is disabled. Changing them only sets up the multiply value for PLL usage in PBE mode
    - c) BLPE: If transitioning through BLPE mode, clear LP (bit 3) in MCGC2 to 0 here to switch to PBE mode
    - d) PBE: Loop until PLLST (bit 5) in MCGSC is set, indicating that the current source for the PLLS clock is the PLL
    - e) PBE: Then loop until LOCK (bit 6) in MCGSC is set, indicating that the PLL has acquired lock
  3. Lastly, PBE mode transitions into PEE mode:
    - a) MCGC1 = 0x18 (%00011000)
      - CLKS (bits 7 and 6) in MCGSC1 set to %00 to select the output of the PLL as the system clock source
      - Loop until CLKST (bits 3 and 2) in MCGSC are %11, indicating that the PLL output is selected to feed MCGOUT in the current clock mode

- b) Now, With an RDIV of divide-by-8, a BDIV of divide-by-1, and a VDIV of multiply-by-32,  
 $MCGOUT = [(8 \text{ MHz} / 8) * 32] / 1 = 32 \text{ MHz}$ , and the bus frequency is  $MCGOUT / 2$ , or 16  
MHz

## Multipurpose Clock Generator (MCG)



**Figure 6-9. Flowchart of FEI to PEE Mode Transition using an 8 MHz crystal**

### 6.5.3.2 Example 2: Moving from PEE to BLPI Mode: Bus Frequency =16 kHz

In this example, the MCG moves through the proper operational modes from PEE mode with an 8MHz crystal configured for an 16 MHz bus frequency (see previous example) to BLPI mode with a 16 kHz bus frequency. First, the code sequence is described. Then a flowchart is included that illustrates the sequence.

1. First, PEE must transition to PBE mode:
  - a)  $\text{MCGC1} = 0x98$  (%10011000)
    - CLKS (bits 7 and 6) set to %10 to switch the system clock source to the external reference clock
  - b) Loop until CLKST (bits 3 and 2) in MCGSC are %10, indicating that the external reference clock is selected to feed MCGOUT
2. Then, PBE must transition directly to FBE mode or first through BLPE mode and then to FBE mode:
  - a) BLPE: If a transition through BLPE mode is desired, first set LP (bit 3) in MCGC2 to 1
  - b) BLPE/FBE:  $\text{MCGC3} = 0x18$  (%00011000)
    - PLLS (bit 6) clear to 0 to select the FLL. At this time, with an RDIV value of %011, the PLL reference divider of 8 is switched to an FLL divider of 256 (see [Table 6-2](#)), resulting in a reference frequency of  $8 \text{ MHz} / 256 = 31.25 \text{ kHz}$ . If RDIV was not previously set to %011 (necessary to achieve required 31.25-39.06 kHz FLL reference frequency with an 8 MHz external source frequency), it must be changed prior to clearing the PLLS bit. In BLPE mode, changing this bit only prepares the MCG for FLL usage in FBE mode. With PLLS = 0, the VDIV value does not matter.
    - DIV32 (bit 4) set to 1 (if previously cleared), automatically switches RDIV bits to the proper reference divider for the FLL clock (divide-by-256)
  - c) BLPE: If transitioning through BLPE mode, clear LP (bit 3) in MCGC2 to 0 here to switch to FBE mode
  - d) FBE: Loop until PLLST (bit 5) in MCGSC is clear, indicating that the current source for the PLLS clock is the FLL
  - e) FBE: Optionally, loop until LOCK (bit 6) in the MCGSC is set, indicating that the FLL has acquired lock. Although the FLL is bypassed in FBE mode, it remains enabled and running.
3. Next, FBE mode transitions into FBI mode:
  - a)  $\text{MCGC1} = 0x5C$  (%01011100)
    - CLKS (bits 7 and 6) in MCGSC1 set to %01 to switch the system clock to the internal reference clock
    - IREFS (bit 2) set to 1 to select the internal reference clock as the reference clock source
    - RDIV (bits 5-3) remain unchanged because the reference divider does not affect the internal reference.
  - b) Loop until IREFST (bit 4) in MCGSC is 1, indicating the internal reference clock has been selected as the reference clock source
  - c) Loop until CLKST (bits 3 and 2) in MCGSC are %01, indicating that the internal reference clock is selected to feed MCGOUT

4. Lastly, FBI transitions into BLPI mode.
  - a) MCGC2 = 0x08 (%00001000)
    - LP (bit 3) in MCGSC is 1
    - RANGE, HGO, EREFS, ERCLKEN, and EREFSTEN bits are ignored when the IREFS bit (bit2) in MCGC is set. They can remain set, or be cleared at this point.

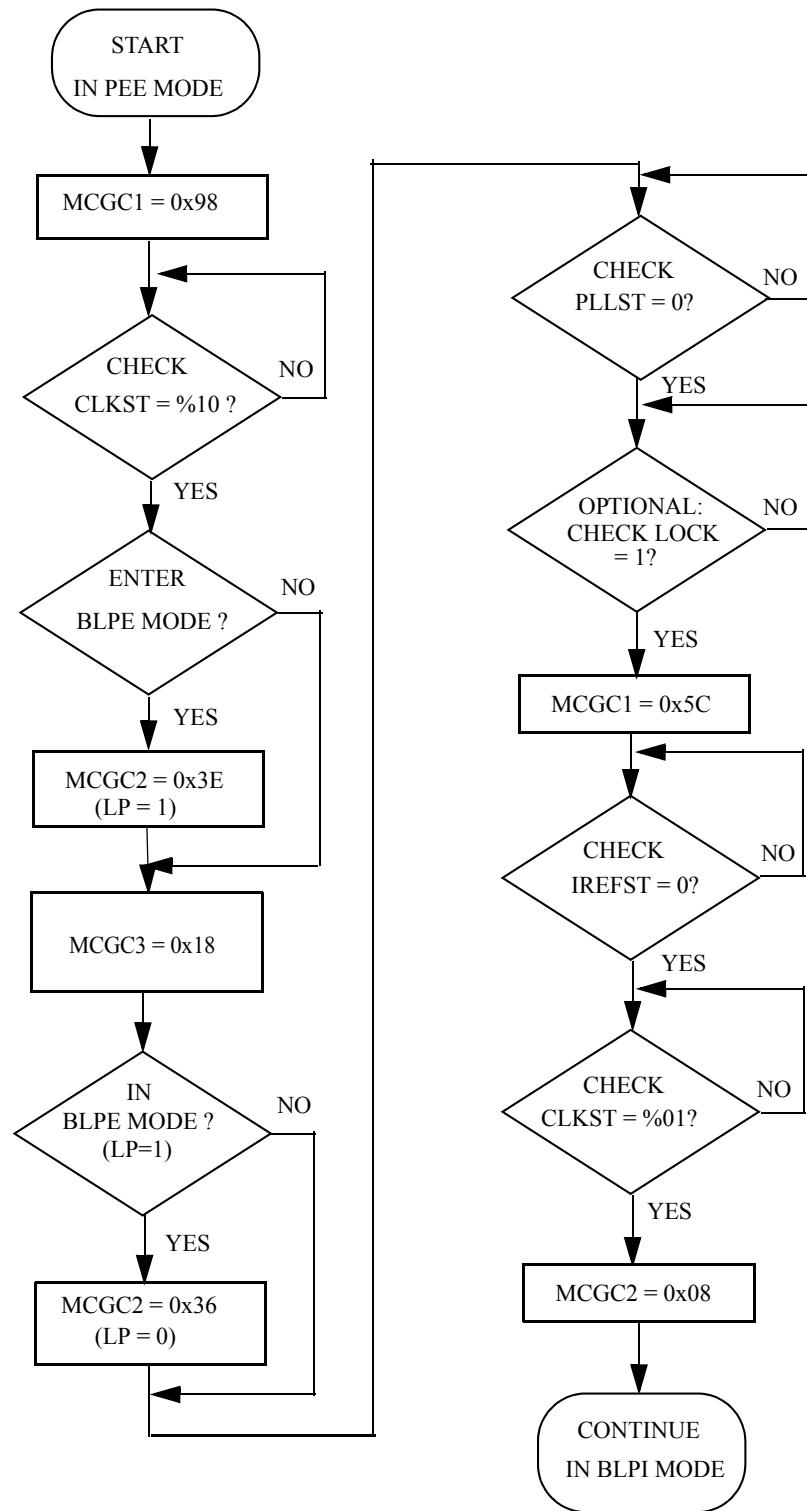


Figure 6-10. Flowchart of PEE to BLPI Mode Transition using an 8 MHz crystal

### 6.5.3.3 Example 3: Moving from BLPI to FEE Mode: External Crystal = 8 MHz, Bus Frequency = 16 MHz

In this example, the MCG moves through the proper operational modes from BLPI mode at a 16 kHz bus frequency running off of the internal reference clock (see previous example) to FEE mode using an 8MHz crystal configured for a 16 MHz bus frequency. First, the code sequence is described. Then a flowchart is included that illustrates the sequence.

1. First, BLPI must transition to FBI mode.
  - a) MCGC2 = 0x00 (%00000000)
    - LP (bit 3) in MCGSC is 0
  - b) Optionally, loop until LOCK (bit 6) in the MCGSC is set, indicating that the FLL has acquired lock. Although the FLL is bypassed in FBI mode, it remains enabled and running.
2. Next, FBI transitions to FEE mode.
  - a) MCGC2 = 0x36 (%00110110)
    - RANGE (bit 5) set to 1 because the frequency of 8 MHz is within the high frequency range
    - HGO (bit 4) set to 1 to configure the crystal oscillator for high gain operation
    - EREFS (bit 2) set to 1, because a crystal is being used
    - ERCLKEN (bit 1) set to 1 to ensure the external reference clock is active
  - b) Loop until OSCINIT (bit 1) in MCGSC is 1, indicating the crystal selected by the EREFS bit has been initialized.
  - c) MCGC1 = 0x18 (%00011000)
    - CLKS (bits 7 and 6) set to %00 to select the output of the FLL as system clock source
    - RDIV (bits 5-3) remain at %011, or divide-by-256 for a reference of 8 MHz / 256 = 31.25 kHz.
    - IREFS (bit 1) cleared to 0, selecting the external reference clock
  - d) Loop until IREFST (bit 4) in MCGSC is 0, indicating the external reference clock is the current source for the reference clock
  - e) Optionally, loop until LOCK (bit 6) in the MCGSC is set, indicating that the FLL has reacquired lock.
  - f) Loop until CLKST (bits 3 and 2) in MCGSC are %00, indicating that the output of the FLL is selected to feed MCGOUT
  - g) Now, with a 31.25 kHz reference frequency, a fixed DCO multiplier of 512, and a bus divider of 1, MCGOUT = 31.25 kHz \* 512 / 1 = 16 MHz. Therefore, the bus frequency is 8 MHz.
  - h) At this point, by default, the DRS[1:0] bits in MCGC4 are set to %00 and DMX32 in MCGC4 is cleared to 0. If a bus frequency of 16MHz is desired instead, set the DRS[1:0] bits to 0x01 to switch the FLL multiplication factor from 512 to 1024 and loop until LOCK (bit 6) in MCGSC is set, indicating that the FLL has reacquired LOCK. To return the bus frequency to 8 MHz, set the DRS[1:0] bits to %00 again, and the FLL multiplication factor switches back to 512. Then loop again until the LOCK bit is set.

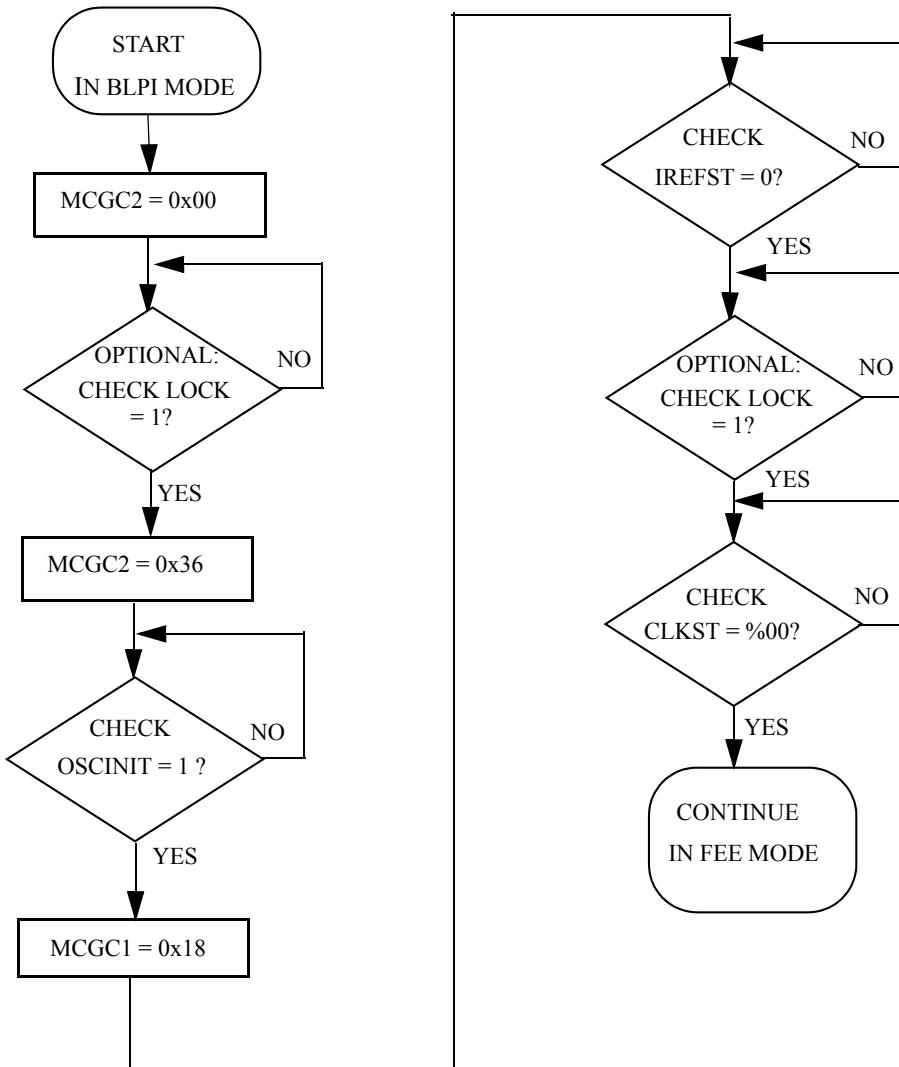


Figure 6-11. Flowchart of BLPI to FEE Mode Transition using an 8 MHz crystal

### 6.5.4 Calibrating the Internal Reference Clock (IRC)

The IRC is calibrated by writing to the MCGTRM register first, then using the FTRIM bit to fine tune the frequency. Refer to this total 9-bit value as the trim value, ranging from 0x000 to 0x1FF, where the FTRIM bit is the LSB.

The trim value after reset is the factory trim value unless the device resets into any BDM mode in which case it is 0x800. Writing a larger value decreases the frequency and smaller values increases the frequency. The trim value is linear with the period, except that slight variations in wafer fab processing produce slight non-linearities between trim value and period. These non-linearities are why an iterative trimming

approach to search for the best trim value is recommended. In example #4 later in this section, this approach is demonstrated.

If a user specified trim value has been found for a device (to replace the factory trim value), this value can be stored in flash memory to save the value. If power is removed from the device, the IRC can easily be re-trimmed to the user specified value by copying the saved value from flash to the MCG registers. Freescale identifies recommended flash locations for storing the trim value for each MCU. Consult the memory map in the data sheet for these locations.

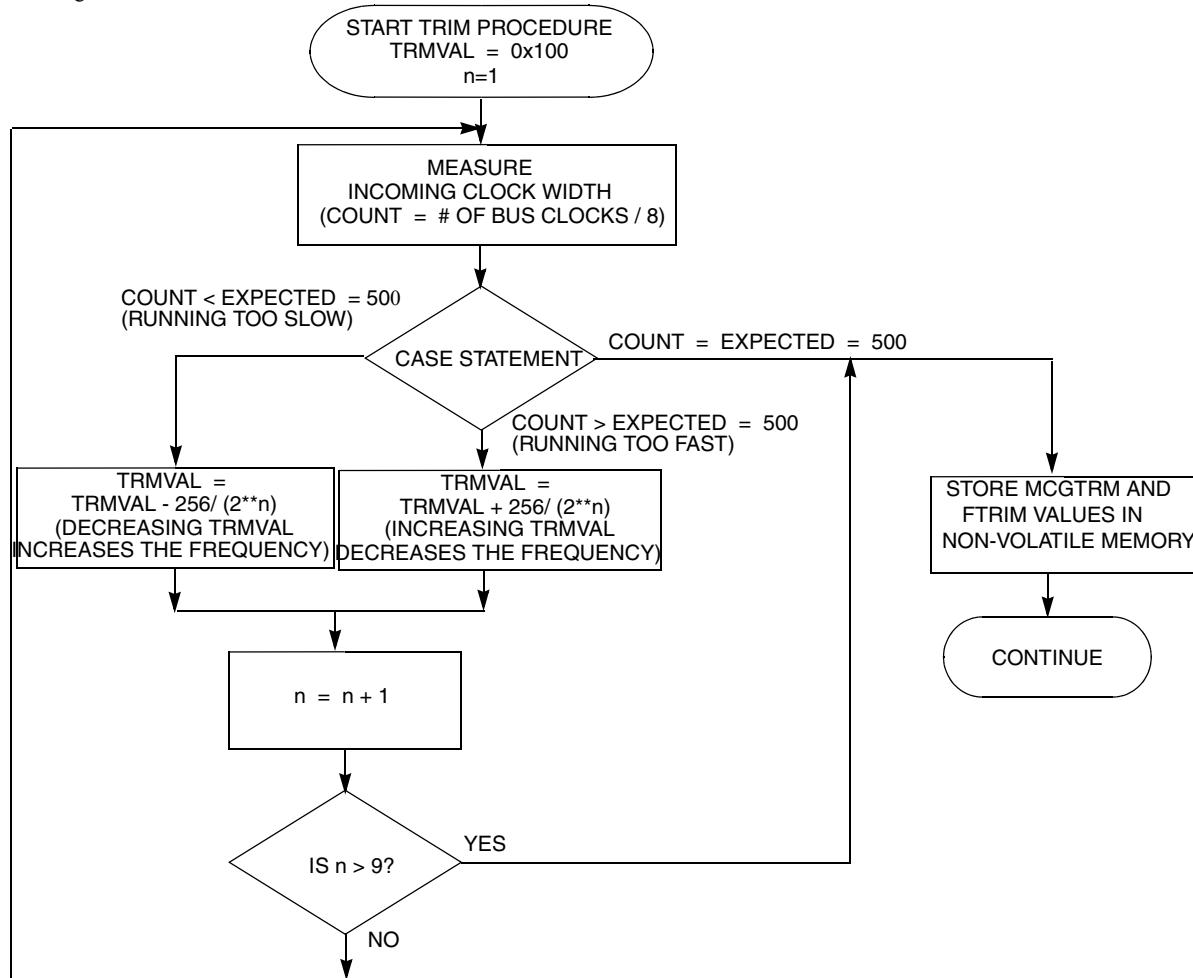
#### 6.5.4.1 Example 4: Internal Reference Clock Trim

For applications that require a user specified tight frequency tolerance, a trimming procedure is provided that allows an accurate internal clock source. This section outlines one example of trimming the internal oscillator. Many other possible trimming procedures are valid and can be used.

In the example below, the MCG trim is calibrated for the 9-bit MCGTRM and FTRIM collective value. This value is referred to as TRMVAL.

Initial conditions:

- 1) Clock supplied from ATE has 500  $\mu$ sec duty period
- 2) MCG configured for internal reference with 8MHz bus



**Figure 6-12. Trim Procedure**

In this particular case, the MCU has been attached to a PCB and the entire assembly is undergoing final test with automated test equipment. A separate signal or message is provided to the MCU operating under user provided software control. The MCU initiates a trim procedure as outlined in [Figure 6-12](#) while the tester supplies a precision reference signal.

If the intended bus frequency is near the maximum allowed for the device, it is recommended to trim using a reference divider value (RDIV setting) of twice the final value. After the trim procedure is complete, the reference divider can be restored. This prevents accidental overshoot of the maximum clock frequency.

# Chapter 7

## ColdFire Core

### 7.1 Introduction

This section describes the organization of the Version 1 (V1) ColdFire® processor core and an overview of the program-visible registers. For detailed information on instructions, see the ISA\_C definition in the *ColdFire Family Programmer's Reference Manual*.

#### 7.1.1 Overview

As with all ColdFire cores, the V1 ColdFire core is comprised of two separate pipelines decoupled by an instruction buffer.

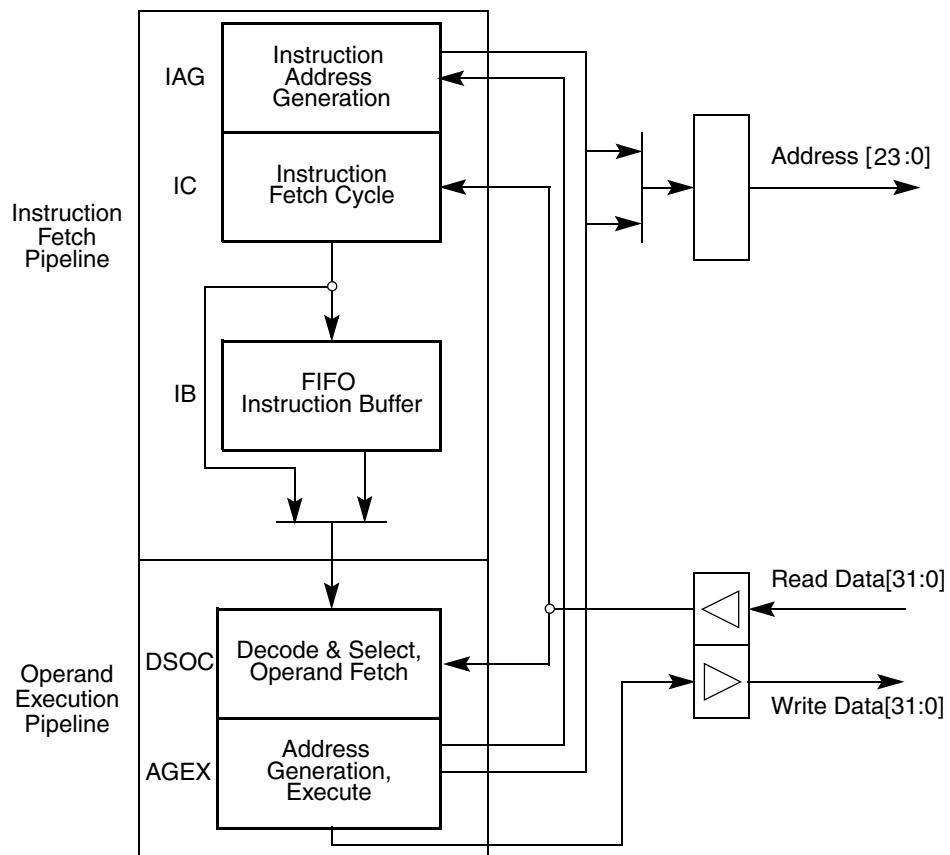


Figure 7-1. V1 ColdFire Core Pipelines

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), that decodes the

instruction, fetches the required operands, and then executes the required function. Because the IFP and OEP pipelines are decoupled by an instruction buffer serving 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 V1 ColdFire core pipeline stages include the following:

- Two-stage instruction fetch pipeline (IFP) (plus optional instruction buffer stage)
  - Instruction address generation (IAG) — Calculates the next prefetch address
  - Instruction fetch cycle (IC)—Initiates prefetch on the processor's local bus
  - Instruction buffer (IB) — Optional buffer stage minimizes fetch latency effects using FIFO queue
- Two-stage operand execution pipeline (OEP)
  - Decode and select/operand fetch cycle (DSOC)—Decodes instructions and fetches the required components for effective address calculation, or the operand fetch cycle
  - Address generation/execute cycle (AGEX)—Calculates operand address or executes the instruction

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 fetched instruction in the IB until it is required by the OEP. The instruction buffer on the V1 core contains three longwords of storage.

For register-to-register and register-to-memory store operations, the instruction passes through both OEP stages once. For memory-to-register and read-modify-write memory operations, an instruction is effectively staged through the OEP twice; the first time to calculate the effective address and initiate the operand fetch on the processor's local bus, and the second time to complete the operand reference and perform the required function defined by the instruction.

The resulting pipeline and local bus structure allow the V1 ColdFire core to deliver sustained high performance across a variety of demanding embedded applications.

## 7.2 Memory Map/Register Description

The following sections describe the processor registers in the user and supervisor programming models. The programming model is selected based on the processor privilege level (user mode or supervisor mode) as defined by the S bit of the status register (SR). [Table 7-1](#) lists the processor registers.

The user-programming model consists 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)

The supervisor programming model is to be used only by system control software 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, that 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 CPU configuration register (CPUCR)

**Table 7-1. ColdFire Core Programming Model**

BDM Command <sup>1</sup>	Register	Width (bits)	Access	Reset Value	Written with MOVEC <sup>2</sup>	Section/Page
<b>Supervisor/User Access Registers</b>						
Load: 0x60 Store: 0x40	Data Register 0 (D0)	32	R/W	See <a href="#">7.3.3.14/7-19</a>	No	<a href="#">7.2.1/7-3</a>
Load: 0x61 Store: 0x41	Data Register 1 (D1)	32	R/W	See <a href="#">7.3.3.14/7-19</a>	No	<a href="#">7.2.1/7-3</a>
Load: 0x6–7 Store: 0x4–7	Data Register –7 (D–D7)	32	R/W	POR: Undefined Else: Unaffected	No	<a href="#">7.2.1/7-3</a>
Load: 0x68–E Store: 0x48–E	Address Register 0–6 (A0–A6)	32	R/W	POR: Undefined Else: Unaffected	No	<a href="#">7.2.2/7-4</a>
Load: 0x6F Store: 0x4F	User A7 Stack Pointer (A7)	32	R/W	POR: Undefined Else: Unaffected	No	<a href="#">7.2.3/7-4</a>
Load: 0xEE Store: 0xCE	Condition Code Register (CCR)	8	R/W	POR: Undefined Else: Unaffected	No	<a href="#">7.2.4/7-5</a>
Load: 0xEF Store: 0xCF	Program Counter (PC)	32	R/W	Contents of location 0x(00)00_0004	No	<a href="#">7.2.5/7-6</a>
<b>Supervisor Access Only Registers</b>						
Load: 0xE0 Store: 0xC0	Supervisor A7 Stack Pointer (OTHER_A7)	32	R/W	Contents of location 0x(00)00_0000	No	<a href="#">7.2.3/7-4</a>
Load: 0xE1 Store: 0xC1	Vector Base Register (VBR)	32	R/W	See section	Yes; Rc = 0x801	<a href="#">7.2.6/7-6</a>
Load: 0xE2 Store: 0xC2	CPU Configuration Register (CPUCR)	32	W	See section	Yes; Rc = 0x802	<a href="#">7.2.7/7-7</a>
Load: 0xEE Store: 0xCE	Status Register (SR)	16	R/W	0x27--	No	<a href="#">7.2.8/7-8</a>

<sup>1</sup> The values listed in this column represent the 8-bit BDM command code used when accessing the core registers via the 1-pin BDM port. For more information see [Chapter 20, “Version 1 ColdFire Debug \(CF1\\_DEBUG\).”](#) (These BDM commands are not similar to other ColdFire processors.)

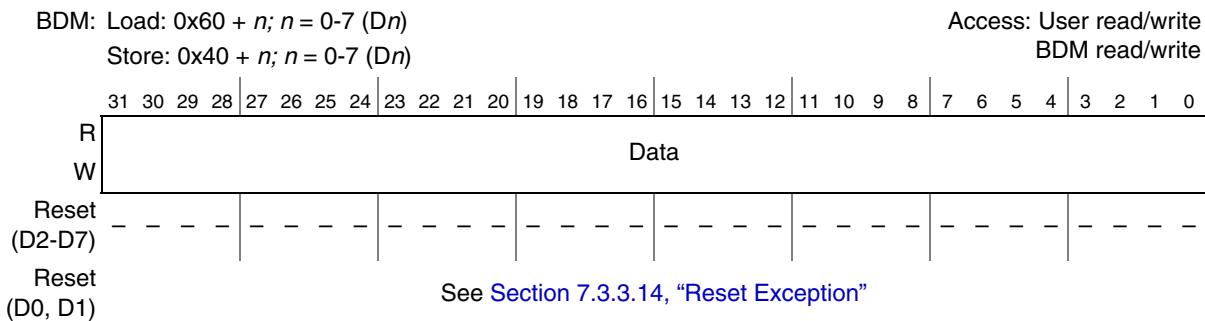
<sup>2</sup> If the given register is written using the MOVEC instruction, the 12-bit control register address (Rc) is also specified.

## 7.2.1 Data Registers (D0–D7)

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

**NOTE**

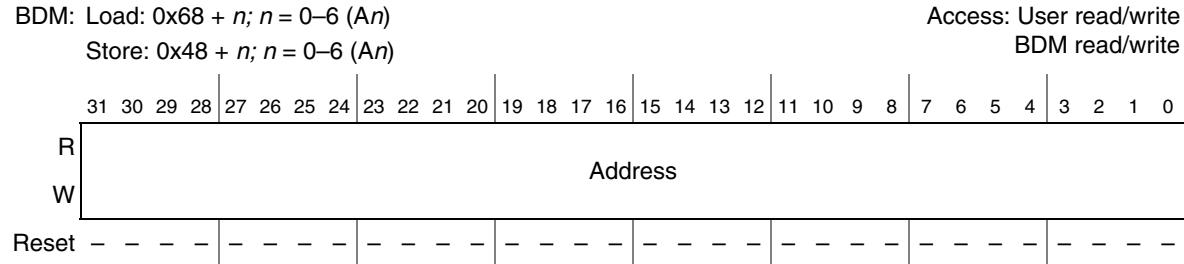
Registers D0 and D1 contain hardware configuration details after reset. See [Section 7.3.3.14, “Reset Exception”](#) for more details.



**Figure 7-2. Data Registers (D0–D7)**

## 7.2.2 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.



**Figure 7-3. Address Registers (A0–A6)**

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

This ColdFire 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 program-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).

To support dual stack pointers, the following two supervisor instructions are included in 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*. All other instruction references to the stack pointer, explicit or implicit, access the active A7 register.

#### NOTE

The SSP is loaded during reset exception processing with the contents of location 0x(00)00\_0000.

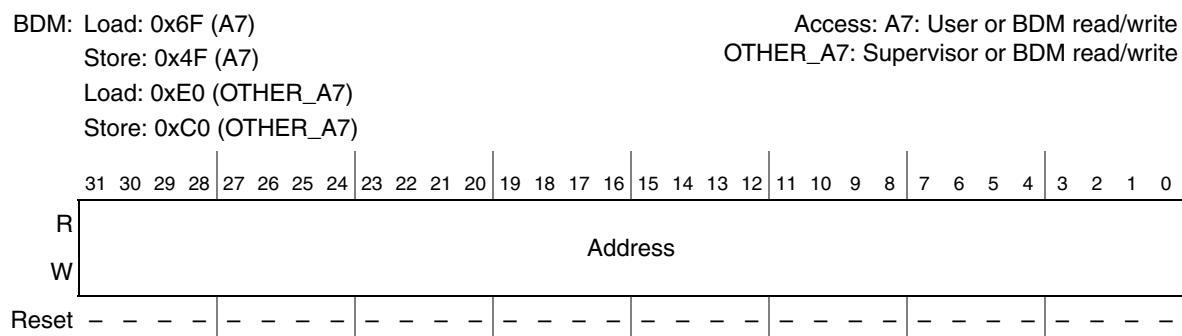


Figure 7-4. Stack Pointer Registers (A7 and OTHER\_A7)

#### 7.2.4 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. The extend bit (X) is also an input operand during multiprecision arithmetic computations. The CCR register must be explicitly loaded after reset and before any compare (CMP), Bcc, or Scc instructions are executed.

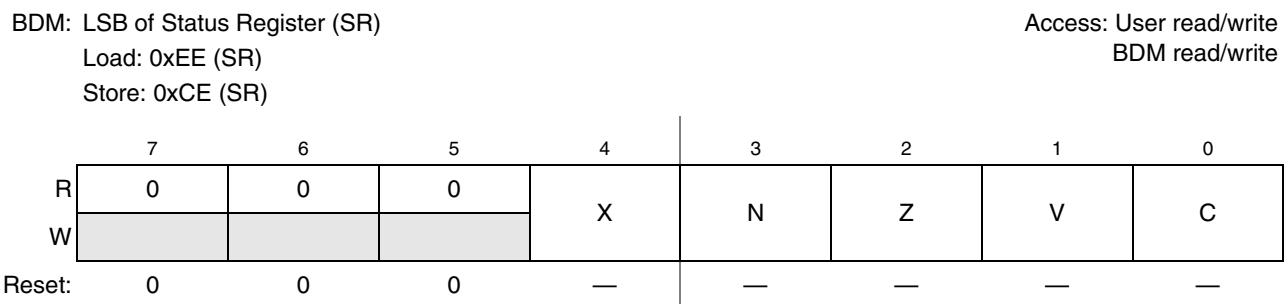


Figure 7-5. Condition Code Register (CCR)

Table 7-2. CCR Field Descriptions

Field	Description
7–5	Reserved, must be cleared.
4 X	Extend condition code bit. Set to the C-bit value for arithmetic operations; otherwise not affected or set to a specified result.

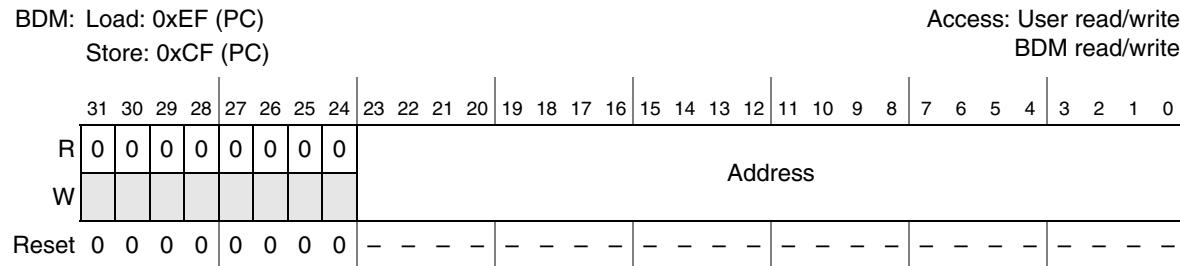
**Table 7-2. CCR Field Descriptions (continued)**

Field	Description
3 N	Negative condition code bit. Set if most significant bit of the result is set; otherwise cleared.
2 Z	Zero condition code bit. Set if result equals zero; otherwise cleared.
1 V	Overflow condition code bit. Set if an arithmetic overflow occurs implying the result cannot be represented in 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.

### **7.2.5 Program Counter (PC)**

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

The PC is initially loaded during reset exception processing with the contents at location 0x0000\_0004.



**Figure 7-6. Program Counter Register (PC)**

### 7.2.6 Vector Base Register (VBR)

The VBR contains the base address of the exception vector table in the 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 MB boundary.

In addition, because the V1 ColdFire core supports a 16 MB address space, the upper byte of the VBR is also forced to zero. The VBR can be used to relocate the exception vector table from its default position in the flash memory (address 0x(00)00 0000) to the base of the RAM (address 0x(00)80 0000) if needed.

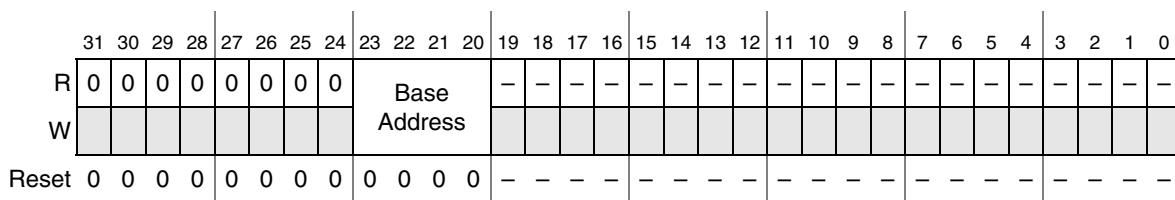
BDM: 0x801 (VBR)

Load: 0xE1 (VBR)

Store: 0xC1 (VBR)

Access: Supervisor read/write

## BDM read/write



**Figure 7-7. Vector Base Register (VBR)**

### 7.2.7 CPU Configuration Register (CPUCR)

The CPUCR provides supervisor mode configurability of specific core functionality. Certain hardware features can be enabled/disabled individually based on the state of the CPUCR.

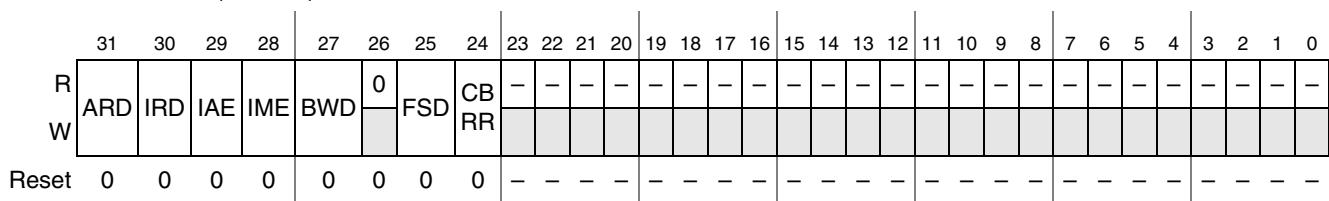
BDM: 0x802 (CPUCR)

Load: 0xE2 (CPUCR)

Store: 0xC2 (CPUCR)

Access: Supervisor read/write

## BDM read/write



**Figure 7-8. CPU Configuration Register (CPUCR)**

**Table 7-3. CPUCR Field Descriptions**

Field	Description
31 ARD	<p>Address-related reset disable. Used to disable the generation of a reset event in response to a processor exception caused by an address error, a bus error, an RTE format error, or a fault-on-fault halt condition.</p> <p>0 The detection of these types of exception conditions or the fault-on-fault halt condition generate a reset event. 1 No reset is generated in response to these exception conditions.</p>
30 IRD	<p>Instruction-related reset disable. Used to disable the generation of a reset event in response to a processor exception caused by the attempted execution of an illegal instruction (except for the ILLEGAL opcode), illegal line A, illegal line F instructions, or a privilege violation.</p> <p>0 The detection of these types of exception conditions generate a reset event. 1 No reset is generated in response to these exception conditions.</p>
29 IAE	<p>Interrupt acknowledge (IACK) enable. Forces the processor to generate an IACK read cycle from the interrupt controller during exception processing to retrieve the vector number of the interrupt request being acknowledged. The processor's execution time for an interrupt exception is slightly improved when this bit is cleared.</p> <p>0 The processor uses the vector number provided by the interrupt controller at the time the request is signaled. 1 IACK read cycle from the interrupt controller is generated.</p>
28 IME	<p>Interrupt mask enable. Forces the processor to raise the interrupt level mask (SR[I]) to 7 during every interrupt exception.</p> <p>0 As part of an interrupt exception, the processor sets SR[I] to the level of the interrupt being serviced. 1 As part of an interrupt exception, the processor sets SR[I] to 7. This disables all level 1-6 interrupt requests but allows recognition of the edge-sensitive level 7 requests.</p>

**Table 7-3. CPUCR Field Descriptions (continued)**

Field	Description
27 BWD	Buffered write disable. The ColdFire core is capable of marking processor memory writes as bufferable or non-bufferable. 0 Writes are buffered and the bus cycle is terminated immediately with zero wait states. 1 Disable the buffering of writes. In this configuration, the write transfer is terminated based on the response time of the addressed destination memory device. <b>Note:</b> If buffered writes are enabled (BWD = 0), any error status is lost as the immediate termination of the data transfer assumes an error-free completion.
26	Reserved, must be cleared.
25 FSD	Flash speculation disabled. Disables certain performance-enhancing features related to address speculation in the flash memory controller. 0 The flash controller tries to speculate on read accesses to improve processor performance by minimizing the exposed flash memory access time. Recall the basic flash access time is two processor cycles. 1 Certain flash address speculation is disabled.
24 CBRR	Crossbar round-robin arbitration enable. Configures the crossbar slave ports to fixed-priority or round-robin arbitration. 0 Fixed-priority arbitration 1 Round robin arbitration
23–0	Reserved, must be cleared.

## 7.2.8 Status Register (SR)

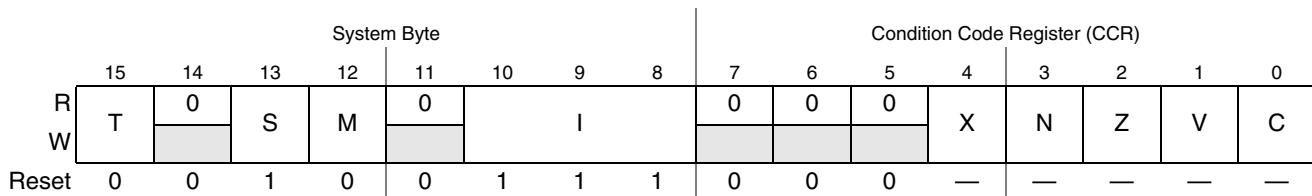
The SR stores the processor status and includes the CCR, the interrupt priority mask, and other control bits. In supervisor mode, software can access the entire SR. In user mode, only the lower 8 bits (CCR) are accessible. The control bits indicate the following states for the processor: trace mode (T 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. The lower byte of the SR (the CCR) must be loaded explicitly after reset and before any compare (CMP), Bcc, or Scc instructions execute.

BDM: Load: 0xEE (SR)

Store: 0xCE (SR)

Access: Supervisor read/write

BDM read/write

**Figure 7-9. Status Register (SR)****Table 7-4. SR Field Descriptions**

Field	Description
15 T	Trace enable. When set, the processor performs a trace exception after every instruction.
14	Reserved, must be cleared.

**Table 7-4. SR Field Descriptions (continued)**

Field	Description
13 S	Supervisor/user state. 0 User mode 1 Supervisor mode
12 M	Master/interrupt state. Bit is cleared by an interrupt exception and software can set it during execution of the RTE or move to SR instructions.
11	Reserved, must be cleared.
10–8 I	Interrupt level mask. Defines current interrupt level. Interrupt requests are inhibited for all priority levels less than or equal to current level, except edge-sensitive level 7 requests, which cannot be masked.
7–0 CCR	Refer to <a href="#">Section 7.2.4, “Condition Code Register (CCR)”. </a>

## 7.3 Functional Description

### 7.3.1 Instruction Set Architecture (ISA\_C)

The original ColdFire instruction set architecture (ISA\_A) was derived from the M68000 family opcodes based on extensive analysis of embedded application code. The ISA was optimized for code compiled from high-level languages where the dominant operand size was the 32-bit integer declaration. This approach minimized processor complexity and cost, while providing excellent performance for compiled applications.

After the initial ColdFire compilers were created, developers noted there were certain ISA additions that would enhance code density and overall performance. Additionally, as users implemented ColdFire-based designs into a wide range of embedded systems, they found certain frequently-used instruction sequences that could be improved by the creation of additional instructions.

The original ISA definition minimized support for instructions referencing byte- and word-sized operands. Full support for the move byte and move word instructions was provided, but the only other opcodes supporting these data types are CLR (clear) and TST (test). A set of instruction enhancements has been implemented in subsequent ISA revisions, ISA\_B and ISA\_C. The new opcodes primarily addressed three areas:

1. Enhanced support for byte and word-sized operands
2. Enhanced support for position-independent code
3. Miscellaneous instruction additions to address new functionality

[Table 7-5](#) summarizes the instructions added to revision ISA\_A to form revision ISA\_C. For more details see the *ColdFire Family Programmer’s Reference Manual*.

**Table 7-5. Instruction Enhancements over Revision ISA\_A**

<b>Instruction</b>	<b>Description</b>
BITREV	The contents of the destination data register are bit-reversed; that is, new Dn[31] equals old Dn[0], new Dn[30] equals old Dn[1], ..., new Dn[0] equals old Dn[31].
BYTEREV	The contents of the destination data register are byte-reversed; that is, new Dn[31:24] equals old Dn[7:0], ..., new Dn[7:0] equals old Dn[31:24].
FF1	The data register, Dn, is scanned, beginning from the most-significant bit (Dn[31]) and ending with the least-significant bit (Dn[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.
MOV3Q.L	Moves 3-bit immediate data to the destination location.
Move from USP	User Stack Pointer → Destination register
Move to USP	Source register → User Stack Pointer
MVS.{B,W}	Sign-extends source operand and moves it to destination register.
MVZ.{B,W}	Zero-fills source operand and moves it to destination register.
SATS.L	Performs saturation operation for signed arithmetic and updates destination register, depending on CCR[V] and bit 31 of the register.
TAS.B	Performs indivisible read-modify-write cycle to test and set addressed memory byte.
Bcc.L	Branch conditionally, longword
BSR.L	Branch to sub-routine, longword
CMP.{B,W}	Compare, byte and word
CMPA.W	Compare address, word
CMPI.{B,W}	Compare immediate, byte and word
MOVEI	Move immediate, byte and word to memory using Ax with displacement
STLDSR	Pushes the contents of the status register onto the stack and then reloads the status register with the immediate data value.

### 7.3.2 Exception Processing Overview

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

- A simplified exception vector table
- Reduced relocation capabilities using the vector-base register
- A single exception stack frame format
- Use of separate system stack pointers for user and supervisor modes.

All ColdFire processors use an instruction restart exception model. Exception processing includes all actions from fault condition detection to the initiation of fetch for first handler instruction. Exception processing is comprised of four major steps:

1. The processor makes an internal copy of the SR and then enters supervisor mode by setting the S bit and disabling trace mode by clearing the T bit. The interrupt exception also forces the M bit to be cleared and the interrupt priority mask to set to current interrupt request level.
2. The processor determines the exception vector number. For all faults except interrupts, the processor performs this calculation based on exception type. For interrupts, the processor performs an interrupt-acknowledge (IACK) bus cycle to obtain the vector number from the interrupt controller if CPUCR[IAE] is set. The IACK cycle is mapped to special locations within the interrupt controller's address space with the interrupt level encoded in the address. If CPUCR[IAE] is cleared, the processor uses the vector number supplied by the interrupt controller at the time the request was signaled for improved performance.
3. The processor saves the current context by creating an exception stack frame on the system stack. The exception stack frame is created at a 0-modulo-4 address on top of the system stack pointed to by the supervisor stack pointer (SSP). As shown in [Figure 7-10](#), 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).
4. The processor calculates the address of the first instruction of the exception handler. By definition, the exception vector table is aligned on a 1 MB 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  $(4 \times \text{vector number})$ . After the exception vector has been fetched, the vector contents determine the address of the first instruction of the desired handler. After the instruction fetch for the first opcode of the handler has 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 MB address boundary (see [Table 7-6](#)). For the V1 ColdFire core, the only practical locations for the vector table are based at 0x(00)00\_0000 in the flash or 0x(00)80\_0000 in the internal SRAM.

The table contains 256 exception vectors; the first 64 are defined for the core and the remaining 192 are device-specific peripheral interrupt vectors. See [Chapter 8, “Interrupt Controller \(CF1\\_INTC\)](#)” for details on the device-specific interrupt sources.

For the V1 ColdFire core, the table is partially populated with the first 64 reserved for internal processor exceptions, while vectors 64–102 are reserved for the peripheral I/O requests and the seven software interrupts. Vectors 103–255 are unused and reserved.

**Table 7-6. Exception Vector Assignments**

Vector Number(s)	Vector Offset (Hex)	Stacked Program Counter	Assignment
0	0x000	—	Initial supervisor stack pointer
1	0x004	—	Initial program counter

**Table 7-6. Exception Vector Assignments (continued)**

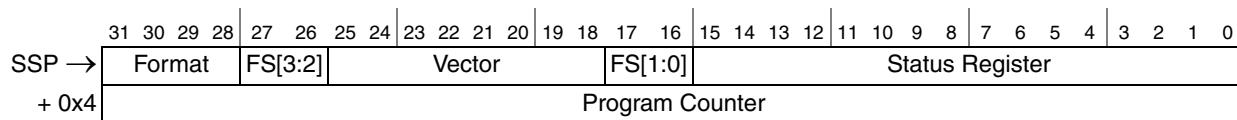
Vector Number(s)	Vector Offset (Hex)	Stacked Program Counter	Assignment
2	0x008	Fault	Access error
3	0x00C	Fault	Address error
4	0x010	Fault	Illegal instruction
5–7	0x014–0x01C	—	Reserved
8	0x020	Fault	Privilege violation
9	0x024	Next	Trace
10	0x028	Fault	Unimplemented line-A opcode
11	0x02C	Fault	Unimplemented 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
48–60	0x0C0–0x0F0	—	Reserved
61	0x0F4	Fault	Unsupported instruction
62–63	0x0F8–0x0FC	—	Reserved
64–102	0x100–0x198	Next	Device-specific interrupts
103–255	0x19C–0x3FC	—	Reserved

<sup>1</sup> Fault refers to the PC of the instruction that caused the exception. Next refers to the PC of the 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 disable interrupts effectively, if necessary, by raising the interrupt mask level contained in the status register. In addition, the ISA\_C architecture includes an 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 that services multiple interrupt requests with different interrupt levels. Finally, the V1 ColdFire core includes the CPUCR[IME] bit that forces the processor to automatically raise the mask level to 7 during the interrupt exception, removing the need for any explicit instruction in the service routine to perform this function. For more details, see *ColdFire Family Programmer's Reference Manual*.

### 7.3.2.1 Exception Stack Frame Definition

Figure 7-10 shows exception stack frame. The first longword 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 7-10. Exception Stack Frame Form**

The 16-bit format/vector word contains three 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 7-7](#).

**Table 7-7. Format Field Encodings**

Original SSP @ Time of Exception, Bits 1:0	SSP @ 1st Instruction of Handler	Format Field
00	Original SSP - 8	0100
01	Original SSP - 9	0101
10	Original SSP - 10	0110
11	Original SSP - 11	0111

- 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 exceptions. See [Table 7-8](#).

**Table 7-8. Fault Status Encodings**

FS[3:0]	Definition
00xx	Reserved
0100	Error on instruction fetch
0101	Reserved
011x	Reserved
1000	Error on operand write
1001	Reserved
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 case of an interrupt. See [Table 7-6](#).

### 7.3.3 Processor Exceptions

#### 7.3.3.1 Access Error Exception

The default operation of the V1 ColdFire processor is the generation of an illegal address reset event if an access error (also known as a bus error) is detected. If CPUCR[ARD] is set, the reset is disabled and a processor exception is generated as detailed below.

The exact processor response to an access error depends on the 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 during instruction prefetches 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 is 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 a MOVEM instruction loading from memory, any registers already updated before the fault occurs contain the operands from memory.

The V1 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.

#### 7.3.3.2 Address Error Exception

The default operation of the V1 ColdFire processor is the generation of an illegal address reset event if an address error is detected. If CPUCR[ARD] equals 1, then the reset is disabled and a processor exception is generated as detailed below.

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

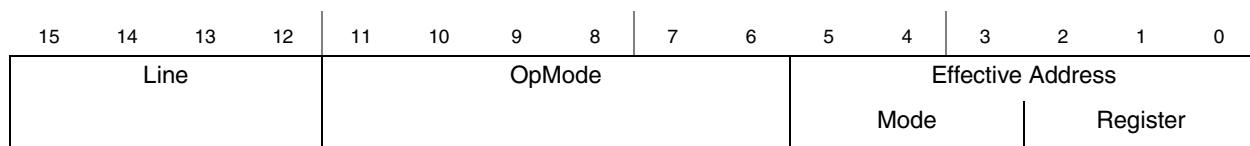
Any attempted use of a word-sized index register (Xn.w) or a scale factor of eight on an indexed effective addressing mode generates an address error, as does an attempted execution of a full-format indexed addressing mode, which is defined by bit 8 of extension word 1 being set.

If an address error occurs on an RTS instruction, the Version 1 ColdFire processor overwrites the faulting return PC with the address error stack frame.

### 7.3.3.3 Illegal Instruction Exception

The default operation of the V1 ColdFire processor is the generation of an illegal opcode reset event if an illegal instruction is detected. If CPUCR[IRD] is set, the reset is disabled and a processor exception is generated as detailed below. There is one special case involving the ILLEGAL opcode (0x4AFC); attempted execution of this instruction always generates an illegal instruction exception, regardless of the state of the CPUCR[IRD] bit.

The ColdFire variable-length instruction set architecture supports three instruction sizes: 16, 32, or 48 bits. The first instruction word is known as the operation word (or opword), while the optional words are known as extension word 1 and extension word 2. The opword is further subdivided into three sections: the upper four bits segment the entire ISA into 16 instruction lines, the next 6 bits define the operation mode (opmode), and the low-order 6 bits define the effective address. See [Figure 7-11](#). The opword line definition is shown in [Table 7-9](#).



**Figure 7-11. ColdFire Instruction Operation Word (Opword) Format**

**Table 7-9. ColdFire Opword Line Definition**

Opword[Line]	Instruction Class
0x0	Bit manipulation, Arithmetic and Logical Immediate
0x1	Move Byte
0x2	Move Long
0x3	Move Word
0x4	Miscellaneous
0x5	Add (ADDQ) and Subtract Quick (SUBQ), Set according to Condition Codes (Scc)
0x6	PC-relative change-of-flow instructions Conditional (Bcc) and unconditional (BRA) branches, subroutine calls (BSR)
0x7	Move Quick (MOVEQ), Move with sign extension (MVS) and zero fill (MVZ)
0x8	Logical OR (OR)
0x9	Subtract (SUB), Subtract Extended (SUBX)
0xA	Move 3-bit Quick (MOV3Q)
0xB	Compare (CMP), Exclusive-OR (EOR)
0xC	Logical AND (AND), Multiply Word (MUL)
0xD	Add (ADD), Add Extended (ADDX)
0xE	Arithmetic and logical shifts (ASL, ASR, LSL, LSR)
0xF	Write DDATA (WDDATA), Write Debug (WDEBUG)

In the original M68000 ISA definition, lines A and F were effectively reserved for user-defined operations (line A) and co-processor instructions (line F). Accordingly, there are two unique exception vectors associated with illegal opwords in these two lines.

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 line-A and most line-F opcodes generate their unique exception types, vector numbers 10 and 11, respectively. ColdFire cores do not provide illegal instruction detection on the extension words on any instruction, including MOVEC.

The V1 ColdFire processor also detects two special cases involving illegal instruction conditions:

1. If execution of the stop instruction is attempted and neither low-power stop nor wait modes are enabled, the processor signals an illegal instruction.
2. If execution of the halt instruction is attempted and BDM is not enabled (XCSR[ENBDM] equals 0), the processor signals an illegal instruction.

In both cases, the processor response is then dependent on the state of CPUCR[IRD]—a reset event or a processor exception.

### 7.3.3.4 Privilege Violation

The default operation of the V1 ColdFire processor is the generation of an illegal opcode reset event if a privilege violation is detected. If CPUCR[IRD] is set, the reset is disabled and a processor exception is generated as detailed below.

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

There is one special case involving the HALT instruction. Normally, this opcode is a supervisor mode instruction, but if the debug module's CSR[UHE] is set, then this instruction can be also be executed in user mode for debugging purposes.

### 7.3.3.5 Trace Exception

To aid in program development, all ColdFire processors provide an instruction-by-instruction tracing capability. While in trace mode, indicated by setting of the SR[T] bit, 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 executes, 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 a TRAP instruction execution while in trace mode. The processor initiates the trap exception and then passes 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[T] in the exception stack frame set) and pass control to the trace handler before returning from the original exception.

### **7.3.3.6 Unimplemented Line-A Opcode**

The default operation of the V1 ColdFire processor is the generation of an illegal opcode reset event if an unimplemented line-A opcode is detected. If CPUCR[IRD] is set, the reset is disabled and a processor exception is generated as detailed below.

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.

### **7.3.3.7 Unimplemented Line-F Opcode**

The default operation of the V1 ColdFire processor is the generation of an illegal opcode reset event if an unimplemented line-F opcode is detected. If CPUCR[IRD] is set, the reset is disabled and a processor exception is generated as detailed below.

A line-F opcode is defined when bits 15-12 of the opword are 0b1111. This exception is generated when attempting to execute an undefined line-F opcode.

### **7.3.3.8 Debug Interrupt**

See [Chapter 20, “Version 1 ColdFire Debug \(CF1\\_DEBUG\),”](#) for a detailed explanation of this exception, which 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). Additionally, SR[M,I] are unaffected by the interrupt.

### **7.3.3.9 RTE and Format Error Exception**

The default operation of the V1 ColdFire processor is the generation of an illegal address reset event if an RTE format error is detected. If CPUCR[ARD] is set, the reset is disabled and a processor exception is generated as detailed below.

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.

### 7.3.3.10 TRAP Instruction Exception

The TRAP #n instruction always forces an exception as part of its execution and is useful for implementing system calls. The TRAP instruction may be used to change from user to supervisor mode.

This set of 16 instructions provides a similar but expanded functionality compared to the S08's SWI (software interrupt) instruction. Do not confuse these instructions and their functionality with the software-scheduled interrupt requests, which are handled like normal I/O interrupt requests by the interrupt controller. The processing of the software-scheduled IRQs can be masked, based on the interrupt priority level defined by the SR[I] field.

### 7.3.3.11 Unsupported Instruction Exception

If execution of a valid instruction is attempted but the required hardware is not present in the processor, an unsupported instruction exception is generated. The instruction functionality can then be emulated in the exception handler, if desired.

All ColdFire cores record the processor hardware configuration in the D0 register immediately after the negation of  $\overline{\text{RESET}}$ . See [Section 7.3.3.14, “Reset Exception,”](#) for details.

For this device, attempted execution of valid integer divide opcodes, CAU, and all MAC and EMAC instructions result in the unsupported instruction exception.

### 7.3.3.12 Interrupt Exception

Interrupt exception processing includes interrupt recognition and the fetch of the appropriate vector from the interrupt controller using an IACK cycle or using the previously-supplied vector number, under control of CPUCR[IAE]. See [Chapter 8, “Interrupt Controller \(CF1\\_INTC\),”](#) for details on the interrupt controller.

### 7.3.3.13 Fault-on-Fault Halt

The default operation of the V1 ColdFire processor is the generation of an illegal address reset event if a fault-on-fault halt condition is detected. If CPUCR[ARD] is set, the reset is disabled and the processor is halted as detailed below.

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 exit this state.

### 7.3.3.14 Reset Exception

Asserting the reset input signal (RESET) 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 SR[S] bit and disables tracing by clearing the SR[T] bit. This exception also clears the SR[M] bit and sets the processor's SR[I] field to the highest level (level 7, 0b111). Next, the VBR is initialized to zero (0x0000\_0000). 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 registers are also affected. Refer to each module in this reference manual for details on these registers.

After the processor is granted the bus, it performs two longword read-bus cycles. The first longword at address 0x(00)00\_0000 is loaded into the supervisor stack pointer and the second longword at address 0x(00)00\_0004 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 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 the BDM to determine the hardware configuration.

Information loaded into D0 defines the processor hardware configuration as shown in [Figure 7-12](#).

BDM: Load: 0x60 (D0)								Access: User read-only BDM read-only								
Store: 0x40 (D0)																
R	PF				VER				REV							
W																
Reset	1	1	0	0	1	1	1	1	0	0	0	1	0	0	0	0
R	MAC				DIV				ISA				DEBUG			
W																
Reset	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	1

Figure 7-12. D0 Hardware Configuration Info

**Table 7-10. D0 Hardware Configuration Info Field Description**

<b>Field</b>	<b>Description</b>
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. Defines the hardware microarchitecture version of ColdFire core. 0001 V1 ColdFire core (This is the value used for this device.) 0010 V2 ColdFire core 0011 V3 ColdFire core 0100 V4 ColdFire core 0101 V5 ColdFire core Else Reserved for future use
19–16 REV	Processor revision number. The default is 0b0000.
15 MAC	MAC present. This bit signals if the optional multiply-accumulate (MAC) execution engine is present in processor core. 0 MAC execute engine not present in core. (This is the value used for this device.) 1 MAC execute engine is present in core.
14 DIV	Divide present. This bit signals if the hardware divider (DIV) is present in the processor core. 0 Divide execute engine not present in core. (This is the value used for this device.) 1 Divide execute engine is present in core.
13–8	Reserved.
7–4 ISA	ISA revision. Defines the instruction-set architecture (ISA) revision level implemented in ColdFire processor core. 0000 ISA_A 0001 ISA_B 0010 ISA_C (This is the value used for this device.) 1000 ISA_A+ Else Reserved
3–0 DEBUG	Debug module revision number. Defines revision level of the debug module used in the ColdFire processor core. 0000 DEBUG_A 0001 DEBUG_B 0010 DEBUG_C 0011 DEBUG_D 0100 DEBUG_E 1001 DEBUG_B+ (This is the value used for this device.) 1011 DEBUG_D+ 1111 DEBUG_D+PST Buffer Else Reserved

Information loaded into D1 defines the local memory hardware configuration as shown in the figure below.

BDM: Load: 0x61 (D1) Store: 0x41 (D1)								Access: User read-only BDM read-only							
31 30 29 28				27 26 25 24				23 22 21 20				19 18 17 16			
R	0	0	0	1	0	0	0	0	FLASHSZ <sup>1</sup>				0	0	0
W															
Reset	0	0	0	1	0	0	0	0	1	0	0	1	0	0	0
R	0	0	0	1	ROMSZ				SRAMSZ				0	0	0
W															
Reset	0	0	0	1	0	0	0	0	0	1	1	1	1	0	0

<sup>1</sup> The FLASHSZ size depends on memory size. The size shown is for 128 KB flash.

Figure 7-13. D1 Hardware Configuration Info

Table 7-11. D1 Hardware Configuration Information Field Description

Field	Description
31–24	Reserved.
18–16	Reserved
15–12	Reserved, resets to 0b0001
11–8 ROMSZ	Boot ROM size. Indicates the size of the boot ROM. 0000 No boot ROM (This is the value used for this device) 0001 512 bytes 0010 1 KB 0011 2 KB 0100 4 KB 0101 8 KB 0110 16 KB 0111 32 KB Else Reserved for future use
7–3 SRAMSZ	SRAM bank size. 00000 No SRAM 00010 512 bytes 00100 1 KB 00110 2 KB 01000 4 KB 01010 8 KB 01100 16 KB 01111 24 KB (This is the value used for this device) 01110 32 KB 10000 64 KB 10010 128 KB Else Reserved for future use
2–0	Reserved.

### 7.3.4 Instruction Execution Timing

This section presents 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.

#### 7.3.4.1 Timing Assumptions

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

1. The 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 IFP to supply opwords and/or extension words.
2. The OEP does not experience any sequence-related pipeline stalls. The most common example 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 clock cycles after the final decode and select/operand fetch cycle (DSOC) of the store instruction. If a subsequent STORE instruction is encountered within this 2-cycle window, it is stalled until the resource again becomes available. Thus, the maximum pipeline stall involving consecutive STORE operations is two 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; for example, 16-bit operands aligned on 0-modulo-2 addresses, 32-bit operands aligned on 0-modulo-4 addresses.

The processor core decomposes misaligned operand references into a series of aligned accesses as shown in [Table 7-12](#).

**Table 7-12. Misaligned Operand References**

address[1:0]	Size	Bus Operations	Additional C(R/W)
01 or 11	Word	Byte, Byte	2(1/0) if read 1(0/1) if write
01 or 11	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

### 7.3.4.2 MOVE Instruction Execution Times

Table 7-13 lists execution times for MOVE.{B,W} instructions; Table 7-14 lists timings for MOVE.L.

#### NOTE

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.

ET with {<ea> = (d16,PC)}	equals ET with {<ea> = (d16,An)}
ET with {<ea> = (d8,PC,Xi*SF)}	equals ET with {<ea> = (d8,An,Xi*SF)}

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

**Table 7-13. MOVE Byte and Word Execution Times**

Source	Destination						
	Rx	(Ax)	(Ax)+	-(Ax)	(d16,Ax)	(d8,Ax,Xi*SF)	xxx.wl
Dy	1(0/0)	1(0/1)	1(0/1)	1(0/1)	1(0/1)	2(0/1)	1(0/1)
Ay	1(0/0)	1(0/1)	1(0/1)	1(0/1)	1(0/1)	2(0/1)	1(0/1)
(Ay)	2(1/0)	3(1/1)	3(1/1)	3(1/1)	3(1/1)	4(1/1))	3(1/1)
(Ay)+	2(1/0)	3(1/1)	3(1/1)	3(1/1)	3(1/1)	4(1/1))	3(1/1)
-(Ay)	2(1/0)	3(1/1)	3(1/1)	3(1/1)	3(1/1)	4(1/1))	3(1/1)
(d16,Ay)	2(1/0)	3(1/1)	3(1/1)	3(1/1)	3(1/1)	—	—
(d8,Ay,Xi*SF)	3(1/0)	4(1/1)	4(1/1)	4(1/1)	—	—	—
xxx.w	2(1/0)	3(1/1)	3(1/1)	3(1/1)	—	—	—
xxx.l	2(1/0)	3(1/1)	3(1/1)	3(1/1)	—	—	—
(d16,PC)	2(1/0)	3(1/1)	3(1/1)	3(1/1)	3(1/1)	—	—
(d8,PC,Xi*SF)	3(1/0)	4(1/1)	4(1/1))	4(1/1))	—	—	—
#xxx	1(0/0)	3(0/1)	3(0/1)	3(0/1)	1(0/1)	—	—

**Table 7-14. MOVE Long Execution Times**

Source	Destination						
	Rx	(Ax)	(Ax)+	-(Ax)	(d16,Ax)	(d8,Ax,Xi*SF)	xxx.wl
Dy	1(0/0)	1(0/1)	1(0/1)	1(0/1)	1(0/1)	2(0/1)	1(0/1)
Ay	1(0/0)	1(0/1)	1(0/1)	1(0/1)	1(0/1)	2(0/1)	1(0/1)
(Ay)	2(1/0)	2(1/1)	2(1/1)	2(1/1)	2(1/1)	3(1/1)	2(1/1)
(Ay)+	2(1/0)	2(1/1)	2(1/1)	2(1/1)	2(1/1)	3(1/1)	2(1/1)
-(Ay)	2(1/0)	2(1/1)	2(1/1)	2(1/1)	2(1/1)	3(1/1)	2(1/1)
(d16,Ay)	2(1/0)	2(1/1)	2(1/1)	2(1/1)	2(1/1)	—	—

**Table 7-14. MOVE Long Execution Times (continued)**

Source	Destination						
	Rx	(Ax)	(Ax)+	-(Ax)	(d16,Ax)	(d8,Ax,Xi*SF)	xxx.wl
(d8,Ay,Xi*SF)	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)	—	—	—
(d16,PC)	2(1/0)	2(1/1)	2(1/1)	2(1/1)	2(1/1)	—	—
(d8,PC,Xi*SF)	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)	—	—	—

**7.3.4.3 Standard One Operand Instruction Execution Times****Table 7-15. One Operand Instruction Execution Times**

Opcode	<EA>	Effective Address							
		Rn	(An)	(An)+	-(An)	(d16,An)	(d8,An,Xn*SF)	xxx.wl	#xxx
BITREV	Dx	1(0/0)	—	—	—	—	—	—	—
BYTEREV	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)	—
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)	—	—	—	—	—	—	—
SATS.L	Dx	1(0/0)	—	—	—	—	—	—	—
SCC	Dx	1(0/0)	—	—	—	—	—	—	—
SWAP	Dx	1(0/0)	—	—	—	—	—	—	—
TAS.B	<ea>	—	3(1/1)	3(1/1)	3(1/1)	3(1/1)	4(1/1)	3(1/1)	—
TST.B	<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)
TST.W	<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)
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)

### 7.3.4.4 Standard Two Operand Instruction Execution Times

Table 7-16. 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)	—	—	—
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(1/0)	3(1/0)	3(1/0)	3(1/0)	4(1/0)	3(1/0)	—
BTST	#imm,<ea>	1(0/0)	3(1/0)	3(1/0)	3(1/0)	3(1/0)	—	—	—
CMP.B	<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)
CMP.W	<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)
CML.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.B	#imm,Dx	1(0/0)	—	—	—	—	—	—	—
CMPI.W	#imm,Dx	1(0/0)	—	—	—	—	—	—	—
CMPI.L	#imm,Dx	1(0/0)	—	—	—	—	—	—	—
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.L	#imm,Dx	—	—	—	—	—	—	—	1(0/0)
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)	—	—	—	—	—	—	—

**Table 7-16. 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
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)	—	—	—	—	—	—	—

### 7.3.4.5 Miscellaneous Instruction Execution Times

**Table 7-17. 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)	—	—	—	—	—	—	—
MOV3QL	#imm,<ea>	1(0/0)	1(0/1)	1(0/1)	1(0/1)	1(0/1)	2(0/1)	1(0/1)	—
MOVE.L	Ay,USP	3(0/0)	—	—	—	—	—	—	—
MOVE.L	USP,Ax	3(0/0)	—	—	—	—	—	—	—
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>,and list	—	1+n(n/0)	—	—	1+n(n/0)	—	—	—
MOVEM.L	and list,<ea>	—	1+n(0/n)	—	—	1+n(0/n)	—	—	—
MVS	<ea>,Dx	1(0/0)	2(1/0)	2(1/0)	2(1/0)	2(1/0)	3(1/0)	2(1/0)	1(0/0)
MVZ	<ea>,Dx	1(0/0)	2(1/0)	2(1/0)	2(1/0)	2(1/0)	3(1/0)	2(1/0)	1(0/0)
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)	—	—	—	—	—	—	—
STLDSR	#imm	—	—	—	—	—	—	—	5(0/1)
STOP	#imm	—	—	—	—	—	—	—	3(0/0) <sup>3</sup>
TRAP	#imm	—	—	—	—	—	—	—	12(1/2)
TPF		1(0/0)	—	—	—	—	—	—	—
TPFW		1(0/0)	—	—	—	—	—	—	—
TPFL		1(0/0)	—	—	—	—	—	—	—
UNLK	Ax	2(1/0)	—	—	—	—	—	—	—

**Table 7-17. Miscellaneous Instruction Execution Times (continued)**

Opcode	<EA>	Effective Address							
		Rn	(An)	(An)+	-(An)	(d16,An)	(d8,An,Xn*SF)	xxx.wl	#xxx
WDDATA	<ea>	—	3(1/0)	3(1/0)	3(1/0)	3(1/0)	4(1/0)	3(1/0)	—
WDEBUG	<ea>	—	5(2/0)	—	—	5(2/0)	—	—	—

<sup>1</sup>The 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] equals 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).

### 7.3.4.6 Branch Instruction Execution Times

**Table 7-18. 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
BRA		—	—	—	—	2(0/1)	—	—	—
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		—	—	7(2/0)	—	—	—	—	—
RTS		—	—	5(1/0)	—	—	—	—	—

**Table 7-19. Bcc Instruction Execution Times**

Opcode	Forward Taken	Forward Not Taken	Backward Taken	Backward Not Taken
Bcc	3(0/0)	1(0/0)	2(0/0)	3(0/0)

# Chapter 8

## Interrupt Controller (CF1\_INTC)

### 8.1 Introduction

The CF1\_INTC interrupt controller (CF1\_INTC) is intended for use in low-cost microcontroller designs using the Version 1 (V1) ColdFire processor core. In keeping with the general philosophy for devices based on this low-end 32-bit processor, the interrupt controller generally supports less programmability compared to similar modules in other ColdFire microcontrollers and embedded microprocessors. However, CF1\_INTC provides the required functionality with a minimal silicon cost.

These requirements guide the CF1\_INTC module definition to support Freescale's Controller Continuum:

- The priorities of the interrupt requests between comparable HCS08 and V1 ColdFire devices are identical.
- Supports a mode of operation (through software convention with hardware assists) equivalent to the S08's interrupt processing with only one level of nesting.
- Leverages the current ColdFire interrupt controller programming model and functionality, but with a minimal hardware implementation and cost.

**Table 8-1** provides a high-level architectural comparison between HCS08 and ColdFire exception processing as these differences are important in the definition of the CF1\_INTC module. Throughout this document, the term IRQ refers to an interrupt request and ISR refers to an interrupt service routine to process an interrupt exception.

**Table 8-1. Exception Processing Comparison**

Attribute	HCS08	V1 ColdFire
Exception Vector Table	32 two-byte entries, fixed location at upper end of memory	103 four-byte entries, located at lower end of memory at reset, relocatable with the VBR
More on Vectors	2 for CPU + 30 for IRQs, reset at upper address	64 for CPU + up to 44 for IRQs, reset at lowest address
Exception Stack Frame	5-byte frame: CCR, A, X, PC	8-byte frame: F/V, SR, PC; General-purpose registers (An, Dn) must be saved/restored by the ISR
Interrupt Levels	$1 = f(CCR[I])$	$7 = f(SR[I])$ with automatic hardware support for nesting
Non-Maskable IRQ Support	No	Yes, with level 7 interrupts
Core-enforced IRQ Sensitivity	No	Level 7 is edge sensitive, else level sensitive
INTC Vectoring	Fixed priorities and vector assignments	Fixed priorities and vector assignments, plus any 2 IRQs can be remapped as the highest priority level 6 requests

**Table 8-1. Exception Processing Comparison (continued)**

Attribute	HCS08	V1 ColdFire
Software IACK	No	Yes
Exit Instruction from ISR	RTI	RTE

## 8.1.1 Overview

Interrupt exception processing includes interrupt recognition, aborting the current instruction execution stream, storing an 8-byte exception stack frame in the memory, calculation of the appropriate vector, and passing control to the specified interrupt service routine.

Unless specifically noted otherwise, all ColdFire processors sample for interrupts once during each instruction's execution during the first cycle of execution in the OEP. Additionally, all ColdFire processors use an instruction restart exception model.

The ColdFire processor architecture defines a 3-bit interrupt priority mask field in the processor's status register (SR[I]). This field, and the associated hardware, support seven levels of interrupt requests with the processor providing automatic nesting capabilities. The levels are defined in descending numeric order with  $7 > 6 \dots > 1$ . Level 7 interrupts are treated as non-maskable, edge-sensitive requests while levels 6–1 are maskable, level-sensitive requests. The SR[I] field defines the processor's current interrupt level. The processor continuously compares the encoded IRQ level from CF1\_INTC against SR[I]. Recall that interrupt requests are inhibited for all levels less than or equal to the current level, except the edge-sensitive level 7 request that cannot be masked.

Exception processing for ColdFire processors is streamlined for performance and includes all actions from detecting the fault condition to the initiation of fetch for the first handler instruction. Exception processing is comprised of four major steps.

1. The processor makes an internal copy of the status register (SR) and enters supervisor mode by setting SR[S] and disabling trace mode by clearing SR[T]. The occurrence of an interrupt exception also forces the master mode (M) bit to clear and the interrupt priority mask (I) to set to the level of the current interrupt request.
2. 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 IACK bus cycle to obtain the vector number from the interrupt controller if CPUCR[IAE] equals 1. The IACK cycle is mapped to special locations within the interrupt controller's IPS address space with the interrupt level encoded in the address. If CPUCR[IAE] equals 0, the processor uses the vector number supplied by the interrupt controller at the time the request was signaled (for improved performance).
3. The processor saves the current context by creating an exception stack frame on the system stack. As a result, exception stack frame is created at a 0-modulo-4 address on top of the system stack defined by the supervisor stack pointer (SSP). The processor uses an 8-byte stack frame for all exceptions. It contains the vector number of the exception, the contents of the status register at the time of the exception, and the program counter (PC) at the time of the exception. 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). For interrupts, the stacked PC is always the address of the next instruction to be executed.

- The processor calculates the address of the first instruction of the exception handler. By definition, the exception vector table is aligned on a 1MB boundary. This instruction address is generated by fetching a 32-bit exception vector from the table located at the address defined in the vector base register (VBR). The index into the exception table is calculated as  $(4 \times \text{vector number})$ . After the exception vector has been fetched, the contents of the vector serve as a 32-bit pointer to 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-MB address boundary. For the V1 ColdFire core, the only practical locations for the vector table are based at 0x(00)00\_0000 in the flash or 0x(00)80\_0000 in the RAM. The table contains 256 exception vectors; the first 64 are reserved for internal processor exceptions, and the remaining 192 are device-specific interrupt vectors. The IRQ assignment table is partially populated depending on the exact set of peripherals for the given device.

The exception vector table for MCF51CN128 series devices is shown in [Table 8-2](#).

**Table 8-2. MC51CN128 Series Exception and Interrupt Vector Table**

Vector Address Offset	Interrupt Level	Priority	Vector Number	Stacked Program Counter	Vector Description	Enable	Source	Vector Name
0x000		N/A	0	—	Initial supervisor stack pointer	N/A	N/A	Vreset
0x004		N/A	1	—	Initial program counter	N/A	N/A	
0x008–0x0FC		N/A	2–63	—	Reserved for internal CPU exceptions (see Table 7-6)			
	7	7-4			Reserved			
0x100	7	mid	64	Next	IRQ_pin	IRQ_SC[IRQIE]	IRQ_SC[IRQF]	Virq
0x104	7	3	65	Next	Low_voltage_detect	PMC_LVDIE	PMC_LVDF	Vlvd
						PMC_LVWIE	PMC_LVWF	
0x108	7	2	66	Next	MCG_lock	MCG_C3[LOLIE]	MCG_SC[LOLS]	Vlol
	7	1			Reserved			
	6	7			Reserved for remapped vector #1			Vl6p7
	6	6			Reserved for remapped vector #2			Vl6p6
0x10C	6	5	67	Next	TPM1_ch0	TPM1_C0SC[CH0IE]	TPM1_C0SC[CH0F ]	Vtpm1ch0

**Table 8-2. MC51CN128 Series Exception and Interrupt Vector Table (continued)**

Vector Address Offset	Interrupt Level	Priority	Vector Number	Stacked Program Counter	Vector Description	Enable	Source	Vector Name
0x110	6	4	68	Next	TPM1_ch1	TPM1_C1SC[CH1IE]	TPM1_C1SC[CH1F ]	Vtpm1ch1
0x114	6	3	69	Next	TPM1_ch2	TPM1_C2SC[CH1IE]	TPM1_C2SC[CH2F ]	Vtpm1ch2
0x118	6	2	70	Next	TPM1_ovfl	TPM1_SC[TOIE]	TPM1_SC[TOF]	Vtpm1ovf
0x11C	6	1	71	Next	MTIM1_ovfl	MTIM1_TOIE	MTIM1_TOF	
0x120	5	7	72	Next	TPM2_ch0	TPM2_C0SC[CH0IE]	TPM2_C0SC[CH0F ]	Vtpm2ch0
0x124	5	6	73	Next	TPM2_ch1	TPM2_C1SC[CH1IE]	TPM2_C1SC[CH1F ]	Vtpm2ch1
0x128	5	5	74	Next	TPM2_ch2	TPM2_C2SC[CH2IE]	TPM2_C2SC[CH2F ]	Vtpm2ch2
0x12C	5	4	75	Next	TPM2_ovfl	TPM2_SC[TOIE]	TPM2_SC[TOF]	Vtpm2ovf
0x130	5	3	76	Next	SPI1	SPI1_C1[SPIE]	SPI1_S[MODF] SPI1_S[SPRF]	Vspi1
						SPI1_C1[SPTIE]	SPI1_S[SPTEF]	
0x134	5	2	77	Next	SPI2	SPI2_C1[SPIE]	SPI2_S[MODF] SPI2_S[SPRF]	Vspi2
						SPI2_C1[SPTIE]	SPI2_S[SPTEF]	
0x138	5	1	78		MTIM2_ovfl	MTIM2_TOF	MTIM2_TOIE	
0x13C	4	7	79	Next	SCI1_err	SCI1_C3[ORIE]	SCI1_S1[OR]	Vsci1err
						SCI1_C3[FEIE]	SCI1_S1[FE]	
						SCI1_C3[NEIE]	SCI1_S1[NF]	
						SCI1_C3[PEIE]	SCI1_S1[PF]	
0x140	4	6	80	Next	SCI1_rx	SCI1_C2[RIE]	SCI1_S1[RDRF]	Vsci1rx
						SCI1_C2[ILIE]	SCI1_S1[IDLE]	
						SCI1_BDH[LBKDIE]	SCI1_S2[LBKDIF]	
						SCI1_BDH[RXEDGIE]	SCI1_S2[RXEDGIF ]	
0x144	4	5	81	Next	SCI1_tx	SCI1_C2[TCIE]	SCI1_S1[TC]	Vsci1tx
						SCI1_C2[TIE]	SCI1_S1[TDRE]	

**Table 8-2. MC51CN128 Series Exception and Interrupt Vector Table (continued)**

Vector Address Offset	Interrupt Level	Priority	Vector Number	Stacked Program Counter	Vector Description	Enable	Source	Vector Name
0x148	4	4	82	Next	SCI2_err	SCI2_C3[ORIE]	SCI2_S1[OR]	Vsci2err
						SCI2_C3[FEIE]	SCI2_S1[FE]	
						SCI2_C3[NEIE]	SCI2_S1[NF]	
						SCI2_C3[PEIE]	SCI2_S1[PF]	
0x14C	4	3	83	Next	SCI2_rx	SCI2_C2[RIE]	SCI2_S1[RDRF]	Vsci2rx
						SCI2_C2[ILIE]	SCI2_S1[IDLE]	
						SCI2_BDH[LBKDIIE]	SCI2_S2[LBKDIF]	
						SCI2_BDH[RXEDGIE]	SCI2_S2[RXEDGIF ]	
0x150	4	2	84	Next	SCI2_tx	SCI2_C2[TCIE]	SCI2_S1[TC]	Vsci2tx
						SCI2_C2[TIE]	SCI2_S1[TDRE]	
0x154	4	1	85	Next	SCI3_OR <sup>1</sup>	See Vectors 100 - 102		Vsci3or
0x158	3	7	86	Next	FEC TXF	FEC_EIMR[TXF]	FEC_EIR[TXF]	
0x15C	3	6	87	Next	FEC RXF	FEC_EIMR[RXF]	FEC_EIR[RXF]	
0x160	3	5	88	Next	FEC Other <sup>2</sup>	See Vectors 88 - 98		
0x164	3	4	89	Next	FEC HBERR	FEC_EIMR[HBERR]	FEC_EIR[HBERR]	
0x168	3	3	90	Next	FEC BABR	FEC_EIMR[BABR]	FEC_EIR[BABR]	
0x16C	3	2	91	Next	FEC BABT	FEC_EIMR[BABT]	FEC_EIR[BABT]	
0x170	3	1	92	Next	FEC GRA	FEC_EIMR[GRA]	FEC_EIR[GRA]	
0x174	2	7	93	Next	FEC TXB	FEC_EIMR[TXB]	FEC_EIR[TXB]	
0x178	2	6	94	Next	FEC RXB	FEC_EIMR[RXB]	FEC_EIR[RXB]	
0x17C	2	5	95	Next	FEC MII	FEC_EIMR[MII]	FEC_EIR[MII]	
0x180	2	4	96	Next	FEC EBERR	FEC_EIMR[EBERR]	FEC_EIR[EBERR]	
0x184	2	3	97	Next	FEC LC	FEC_EIMR[LC]	FEC_EIR[LC]	
0x188	2	2	98	Next	FEC RL	FEC_EIMR[RL]	FEC_EIR[RL]	
0x18C	2	1	99	Next	FEC UN	FEC_EIMR[UN]	FEC_EIR[UN]	
0x190	1	7	100	Next	SCI3_err	SCI3_C3[ORIE]	SCI3_S1[OR]	Vsci3err
						SCI3_C3[FEIE]	SCI3_S1[FE]	
						SCI3_C3[NEIE]	SCI3_S1[NF]	
						SCI3_C3[PEIE]	SCI3_S1[PF]	

**Table 8-2. MC51CN128 Series Exception and Interrupt Vector Table (continued)**

Vector Address Offset	Interrupt Level	Priority	Vector Number	Stacked Program Counter	Vector Description	Enable	Source	Vector Name
0x194	1	6	101	Next	SCI3_rx	SCI3_C2[RIE]	SCI3_S1[RDRF]	Vsci3rx
						SCI3_C2[ILIE]	SCI3_S1[IDLE]	
						SCI3_BDH[LBKDIIE]	SCI3_S2[LBKDIF]	
						SCI3_BDH[RXEDGIE]	SCI3_S2[RXEDGIF ]	
0x198	1	5	102	Next	SCI3_tx	SCI3_C2[TCIE]	SCI3_S1[TC]	Vsci3tx
						SCI3_C2[TIE]	SCI3_S1[TDRE]	
0x19C	7	0	103	Next	Level 7 Software Interrupt			Force_Lvl7
0x1A0	6	0	104	Next	Level 6 Software Interrupt			Force_Lvl6
0x1A4	5	0	105	Next	Level 5 Software Interrupt			Force_Lvl5
0x1A8	4	0	106	Next	Level 4 Software Interrupt			Force_Lvl4
0x1AC	3	0	107	Next	Level 3 Software Interrupt			Force_Lvl3
0x1B0	2	0	108	Next	Level 2 Software Interrupt			Force_Lvl2
0x1B4	1	0	109	Next	Level 1 Software Interrupt			Force_Lvl1
0x1B8	1	4	110	Next	IIC1	IIC1_CR1[IICIE]	IIC1_SR[IICIF]	Viic1
0x1BC	1	mid	111	Next	IIC2	IIC2_CR1[IICIE]	IIC2_SR[IICIF]	Viic2
0x1C0	1	3	112	Next	ADC1	ADC_SC1[AIEN]	ADC_SC1[COCO]	Vadc
0x1C4	1	2	113	Next	KBI1 and KBI2	KBI1_SC[KBIE] KBI2_SC[KBIE]	KBI1_SC[KBF] KBI2_SC[KBF]	Vkeyboard
0x1C8	1	1	114	Next	RTC	RTC_SC[RTIE]	RTC_SC[RTIF]	Vrtc
0x1CC – 0x3FC	N/A	N/A	115 – 255	Next	Reserved			

<sup>1</sup> The SCI3\_OR interrupt is a logical ORing of SCI3\_rx, SCI3\_tx and SCI3\_err. You can use this vector for SCI3 interrupts to be processed at priority level 4. Use of SCI3\_OR is mutually exclusive with use of the other SCI3 vectors.

<sup>2</sup> The “FEC Other” interrupt is a logical ORing of the FEC HBERR, BABR, BABT, GRA, TXB, RXB, MII, EBERR, LC, RL and UN interrupts. You can use this vector to assign all FEC interrupts at priority level 3. Use of “FEC Other” is mutually exclusive with the use of the interrupt signals used to generate it.

**Table 8-3. Unsupported FTSR Interrupts**

Interrupt	FTSR_Local Enable	FTSR_Source	Description
FTSR_emptybuf	FTSR_FCNFG[CBEIE]	FTSR_FSTAT[FCBEF]	Flash Command Buffer Empty Interrupt
FTSR_coco	FTSR_FCNFG[CCIE]	FTSR_FSTAT[FCCF]	Flash Command Complete Interrupt

The basic ColdFire interrupt controller supports up to 63 request sources mapped as nine priorities for each of the seven supported levels (7 levels  $\times$  9 priorities per level). Within the nine priorities within a level, the mid-point is typically reserved for package-level IRQ inputs. The levels and priorities within the level follow a descending order: 7 > 6 > ... > 1 > 0.

The HCS08 architecture supports a 32-entry exception vector table: the first two vectors are reserved for internal CPU/system exceptions and the remaining are available for I/O interrupt requests. The requirement for an exact match between the interrupt requests and priorities across two architectures means the sources are mapped to a sparsely-populated two-dimensional ColdFire array of seven interrupt levels and nine priorities within the level. The following association between the HCS08 and ColdFire vector numbers applies:

$$\text{ColdFire Vector Number} = 62 + \text{HCS08 Vector Number}$$

The CF1\_INTC performs a cycle-by-cycle evaluation of the active requests and signals the highest-level, highest-priority request to the V1 ColdFire core in the form of an encoded interrupt level and the exception vector associated with the request. The module also includes a byte-wide interface to access its programming model. These interfaces are shown in the simplified block diagram of [Figure 8-1](#).

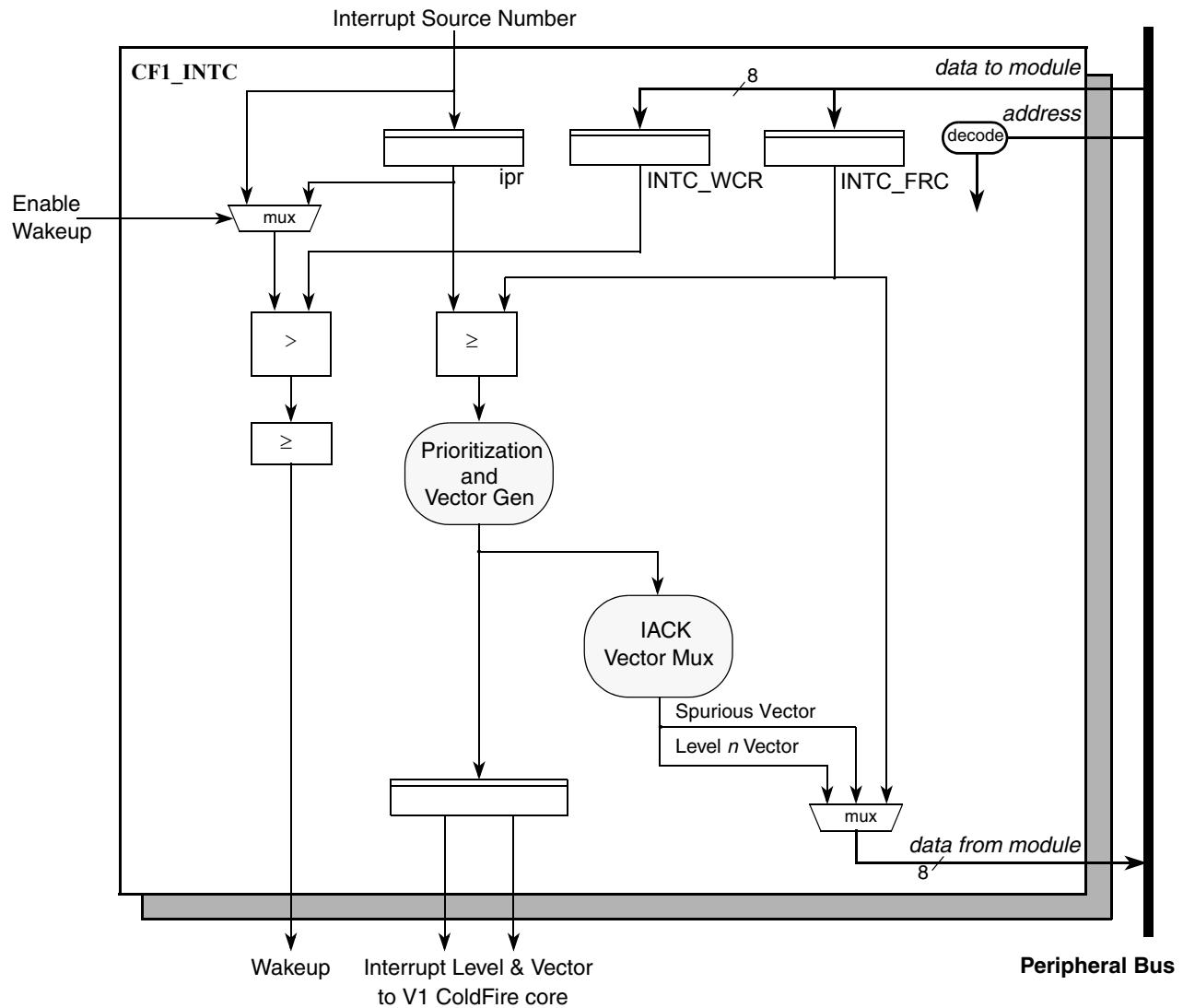


Figure 8-1. CF1\_INTC Block Diagram

### 8.1.2 Features

The Version 1 ColdFire interrupt controller includes:

- Memory-mapped off-platform slave module
  - 64-byte space located at top end of memory: 0x(FF)FF\_FFC0–0x(FF)FF\_FFFF
  - Programming model accessed via the peripheral bus
  - Encoded interrupt level and vector sent directly to processor core
- Support of 44 peripheral I/O interrupt requests plus seven software (one per level) interrupt requests
- Fixed association between interrupt request source and level plus priority
  - 44 I/O requests assigned across seven available levels and nine priorities per level

- Exactly matches HCS08 interrupt request priorities
- Up to two requests can be remapped to the highest maskable level + priority
- Unique vector number for each interrupt source
  - ColdFire vector number = 62 + HCS08 vector number
  - Details on IRQ and vector assignments are device-specific
- Support for service routine interrupt acknowledge (software IACK) read cycles for improved system performance
- Combinatorial path provides wakeup signal from wait mode

### 8.1.3 Modes of Operation

The CF1\_INTC module does not support any special modes of operation. As a memory-mapped slave peripheral located on the platform's slave bus, it responds based strictly on the memory addresses of the connected bus.

One special behavior of the CF1\_INTC deserves mention. When the device enters a wait mode and certain clocks are disabled, there is an input signal that can be asserted to enable a purely-combinational logic path for monitoring the assertion of an interrupt request. After a request of unmasked level is asserted, this combinational logic path asserts an output signal that is sent to the clock generation logic to re-enable the internal device clocks to exit the low-power mode.

## 8.2 External Signal Description

The CF1\_INTC module does not include any external interfaces.

## 8.3 Memory Map/Register Definition

The CF1\_INTC module provides a 64-byte programming model mapped to the upper region of the 16 MB address space. All the register names are prefixed with INTC\_ as an abbreviation for the full module name.

The programming model is referenced using 8-bit accesses. Attempted references to undefined (reserved) addresses or with a non-supported access type (for example, a write to a read-only register) generate a bus error termination.

The programming model follows the definition from previous ColdFire interrupt controllers. This compatibility accounts for the various memory holes in this module's memory map.

The CF1\_INTC module is based at address 0x(FF)FF\_FFC0 (referred to as CF1\_INTC\_BASE throughout the chapter) and occupies the upper 64 bytes of the peripheral space. The module memory map is shown in [Table 8-4](#).

**Table 8-4. CF1\_INTC Memory Map**

Offset Address	Register Name	Register Description	Width (bits)	Access	Reset Value	Section/ Page
0x0C	INTC_ORMR	CF1_INTC OR Mask Register	16	R/W	0x0140	<a href="#">8.3.1/8-10</a>
0x10	INTC_FRC	CF1_INTC Force Interrupt Register	8	R/W	0x00	<a href="#">8.3.2/8-11</a>
0x18	INTC_PL6P7	CF1_INTC Programmable Level 6, Priority 7	8	R/W	0x00	<a href="#">8.3.3/8-12</a>
0x19	INTC_PL6P6	CF1_INTC Programmable Level 6, Priority 6	8	R/W	0x00	<a href="#">8.3.3/8-12</a>
0x1B	INTC_WCR	CF1_INTC Wakeup Control Register	8	R/W	0x80	<a href="#">8.3.4/8-13</a>
0x1E	INTC_SFRC	CF1_INTC Set Interrupt Force Register	8	Write	—	<a href="#">8.3.5/8-14</a>
0x1F	INTC_CFRC	CF1_INTC Clear Interrupt Force Register	8	Write	—	<a href="#">8.3.6/8-15</a>
0x20	INTC_SWIACK	CF1_INTC Software Interrupt Acknowledge	8	Read	0x00	<a href="#">8.3.7/8-16</a>
0x24	INTC_LVL1IACK	CF1_INTC Level 1 Interrupt Acknowledge	8	Read	0x18	<a href="#">8.3.7/8-16</a>
0x28	INTC_LVL2IACK	CF1_INTC Level 2 Interrupt Acknowledge	8	Read	0x18	<a href="#">8.3.7/8-16</a>
0x2C	INTC_LVL3IACK	CF1_INTC Level 3 Interrupt Acknowledge	8	Read	0x18	<a href="#">8.3.7/8-16</a>
0x30	INTC_LVL4IACK	CF1_INTC Level 4 Interrupt Acknowledge	8	Read	0x18	<a href="#">8.3.7/8-16</a>
0x34	INTC_LVL5IACK	CF1_INTC Level 5 Interrupt Acknowledge	8	Read	0x18	<a href="#">8.3.7/8-16</a>
0x38	INTC_LVL6IACK	CF1_INTC Level 6 Interrupt Acknowledge	8	Read	0x18	<a href="#">8.3.7/8-16</a>
0x3C	INTC_LVL7IACK	CF1_INTC Level 7 Interrupt Acknowledge	8	Read	0x18	<a href="#">8.3.7/8-16</a>

### 8.3.1 Interrupt OR Mask Register (INTC\_ORMR)

The INTC\_ORMR provides a mechanism to enable/disable the combined interrupt requests associated with SCI3 and the FEC.

The SCI3 and FEC modules have interrupt sources that are the logical summation (the boolean OR) of multiple interrupt requests. These two interrupt requests can be configured to enable the OR'd version of the interrupt request or the separate individual requests.

The reset state of the INTC\_ORMR disables (masks) the OR'd interrupt requests. If the OR'd version of the SCI3 or FEC interrupt sources is used, they first must be properly enabled (unmasked) by writing to this register.

Offset: CF1\_INTC\_BASE + 0x0C (INTC\_ORMR)

Access: Read/Write

	15	14	13	12	11	10	9	8
R	0	0	0	0	0	0	0	FECDO
W								
Reset	0	0	0	0	0	0	0	1
	7	6	5	4	3	2	1	0
R	0	0	SCI3DO	0	0	0	0	0
W								
Reset	0	0	1	0	0	0	0	0

Figure 8-2. Interrupt OR Mask Register (INTC\_ORMR)

Table 8-5. INTC\_ORMR Register Descriptions

Field	Description
15-9	Reserved, must be cleared.
8 FECDO	Disable (mask) FEC_Other interrupt request 0 FEC_Other interrupt request is enabled; all individual FEC interrupt requests except FEC_TXF and FEC_RXF are disabled 1 FEC_Other interrupt request is disabled; all individual FEC interrupt requests are enabled
7-6	Reserved, must be cleared.
5 SCI3DO	Disable (mask) SCI3_OR interrupt request 0 SCI3_OR request is enabled; all individual SCI3 interrupt requests SCI3_{err, tx, rx} are disabled 1 SCI3_OR request is disabled; all individual SCI3 interrupt requests SCI3_{err, tx, rx} are enabled
4-0	Reserved, must be cleared.

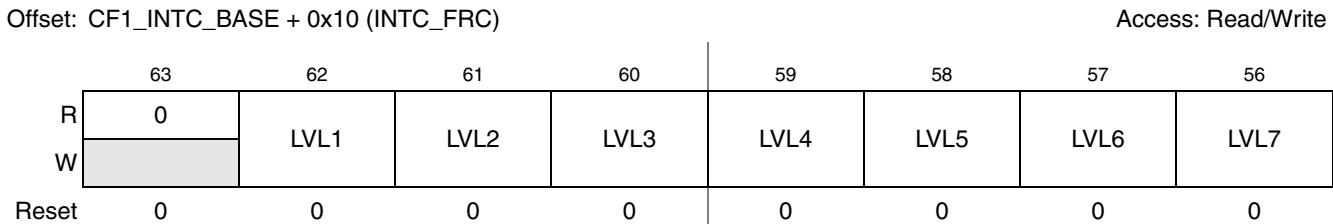
### 8.3.2 Force Interrupt Register (INTC\_FRC)

The INTC\_FRC register allows software to generate a unique interrupt for each possible level at the lowest priority within the level for functional or debug purposes. These interrupts may be self-scheduled by setting one or more of the bits in the INTC\_FRC register. In some cases, the handling of a normal interrupt request may cause critical processing by the service routine along with the scheduling (using the INTC\_FRC register) of a lower priority level interrupt request to be processed at a later time for less-critical task handling.

The INTC\_FRC register may be modified directly using a read-modify-write sequence or through a simple write operation using the set/clear force interrupt registers (INTC\_SFRC, INTC\_CFRC).

#### NOTE

Take special notice of the bit numbers within this register, 63–56. This is for compatibility with other ColdFire interrupt controllers.

**Figure 8-3. Force Interrupt Register (INTC\_FRC)****Table 8-6. INTC\_FRC Field Descriptions**

Field	Description
63	Reserved, must be cleared.
62 LVL1	Force Level 1 interrupt. 0 Negates the forced level 1 interrupt request. 1 Forces a level 1 interrupt request.
61 LVL2	Force Level 2 interrupt. 0 Negates the forced level 2 interrupt request. 1 Forces a level 2 interrupt request.
60 LVL3	Force Level 3 interrupt. 0 Negates the forced level 3 interrupt request. 1 Forces a level 3 interrupt request.
59 LVL4	Force Level 4 interrupt. 0 Negates the forced level 4 interrupt request. 1 Forces a level 4 interrupt request.
58 LVL5	Force Level 5 interrupt. 0 Negates the forced level 5 interrupt request. 1 Forces a level 5 interrupt request.
57 LVL6	Force Level 6 interrupt. 0 Negates the forced level 6 interrupt request. 1 Forces a level 6 interrupt request.
55 LVL7	Force Level 7 interrupt. 0 Negates the forced level 7 interrupt request. 1 Forces a level 7 interrupt request.

### 8.3.3 INTC Programmable Level 6, Priority {7,6} Registers (INTC\_PL6P{7,6})

The level seven interrupt requests cannot have their levels reassigned. However, any of the remaining peripheral interrupt requests can be reassigned as the highest priority maskable requests using these two registers (INTC\_PL6P7 and INTC\_PL6P6). The vector number associated with the interrupt requests does not change. Rather, only the interrupt request's level and priority are altered, based on the contents of the INTC\_PL6P{7,6} registers.

## NOTE

The requests associated with the INTC\_FRC register have a fixed level and priority that cannot be altered.

The INTC\_PL6P7 register specifies the highest-priority, maskable interrupt request that is defined as the level six, priority seven request. The INTC\_PL6P6 register specifies the second-highest-priority, maskable interrupt request defined as the level six, priority six request. Reset clears both registers, disabling any request re-mapping.

For an example of the use of these registers, see [Section 8.6.2, “Using INTC\\_PL6P{7,6} Registers.”](#)

Offset: CF1_INTC_BASE + 0x18 (INTC_PL6P7)	Access: Read/Write																
CF1_INTC_BASE + 0x19 (INTC_PL6P6)																	
R	7        6        5        4                 3        2        1        0																
W	<table border="1" style="width: 100%;"><tr><td>0</td><td>0</td><td colspan="6" style="text-align: right;">REQN</td></tr><tr><td></td><td></td><td colspan="6"></td></tr></table>	0	0	REQN													
0	0	REQN															
Reset	0        0        0        0                 0        0        0        0																

**Figure 8-4. Programmable Level 6, Priority {7,6} Registers (INTC\_PL6P{7,6})**

**Table 8-7. INTC\_PL6P{7,6} Field Descriptions**

Field	Description
7–6	Reserved, must be cleared.
5–0 REQN	Request number. Defines the peripheral IRQ number to be remapped as the level 6, priority 7 (for INTC_PL6P7) request and level 6, priority 6 (for INTC_PL6P6) request. <b>Note:</b> The value must be a valid interrupt number. Unused or reserved interrupt numbers are ignored.

### 8.3.4 INTC Wakeup Control Register (INTC\_WCR)

The interrupt controller provides a combinatorial logic path to generate a special wakeup signal to exit from the wait mode. The INTC\_WCR register defines wakeup condition for interrupt recognition during wait mode. This mode of operation works as follows:

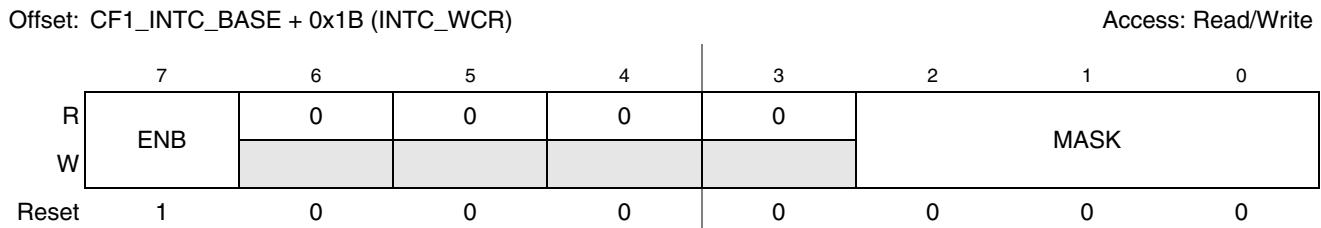
1. Write to the INTC\_WCR to enable this operation (set INTC\_WCR[ENB]) and define the interrupt mask level needed to force the core to exit wait mode (INTC\_WCR[MASK]). The maximum value of INTC\_WCR[MASK] is 0x6 (0b110). The INTC\_WCR is enabled with a mask level of 0 as the default after reset.
2. Execute a stop instruction to place the processor into wait mode.
3. After the processor is stopped, the interrupt controller enables special logic that evaluates the incoming interrupt sources in a purely combinatorial path; no clocked storage elements are involved.
4. If an active interrupt request is asserted and the resulting interrupt level is greater than the mask value contained in INTC\_WCR[MASK], the interrupt controller asserts the wakeup output signal. This signal is routed to the clock generation logic to exit the low-power mode and resume processing.

## Interrupt Controller (CF1\_INTC)

Typically, the interrupt mask level loaded into the processor's status register field (SR[I]) during the execution of the stop instruction matches the INTC\_WCR[MASK] value.

The interrupt controller's wait mode wakeup signal is defined as:

```
wait wakeup = INTC_WCR[ENB] & (level of any asserted int request > INTC_WCR[MASK])
```



**Figure 8-5. Wakeup Control Register (INTC\_WCR)**

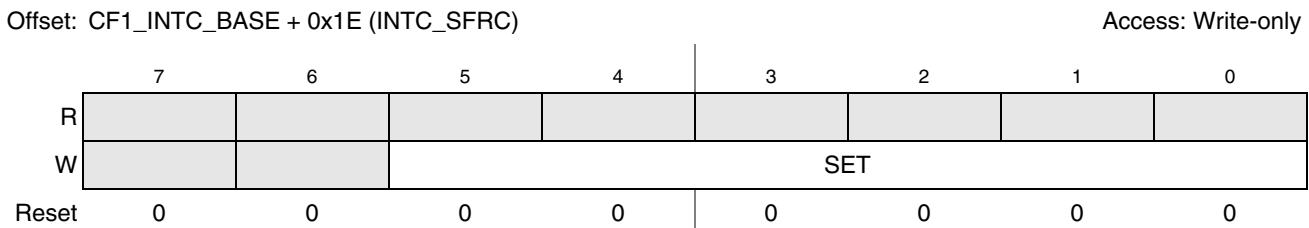
**Table 8-8. INTC\_WCR Field Descriptions**

Field	Description
7 ENB	Enable wakeup signal. 0 Wakeup signal disabled 1 Enables the assertion of the combinational wakeup signal to the clock generation logic.
6–3	Reserved, must be cleared.
2–0 MASK	Interrupt mask level. Defines the interrupt mask level during wait mode and is enforced by the hardware to be within the range 0–6. If INTC_WCR[ENB] is set, when an interrupt request of a level higher than MASK occurs, the interrupt controller asserts the wait mode wakeup signal to the clock generation logic.

### 8.3.5 INTC Set Interrupt Force Register (INTC\_SFRC)

The INTC\_SFRC register provides a simple memory-mapped mechanism to set a given bit in the INTC\_FRC register to assert a specific level interrupt request. The data value written causes the appropriate bit in the INTC\_FRC register to be set. Attempted reads of this register generate an error termination.

This register is provided so interrupt service routines can generate a forced interrupt request without the need to perform a read-modify-write sequence on the INTC\_FRC register.



**Figure 8-6. INTC\_SFRC Register**

**Table 8-9. INTC\_SFRC Field Descriptions**

Field	Description
7–6	Reserved, must be cleared.
5–0 SET	For data values within the 56–62 range, the corresponding bit in the INTC_FRC register is set, as defined below. 0x38 Bit 56, INTC_FRC[LVL7] is set 0x39 Bit 57, INTC_FRC[LVL6] is set 0x3A Bit 58, INTC_FRC[LVL5] is set 0x3B Bit 59, INTC_FRC[LVL4] is set 0x3C Bit 60, INTC_FRC[LVL3] is set 0x3D Bit 61, INTC_FRC[LVL2] is set 0x3E Bit 62, INTC_FRC[LVL1] is set <b>Note:</b> Data values outside this range do not affect the INTC_FRC register. It is recommended the data values be restricted to the 0x38–0x3E (56–62) range to ensure compatibility with future devices.

### 8.3.6 INTC Clear Interrupt Force Register (INTC\_CFRC)

The INTC\_CFRC register provides a simple memory-mapped mechanism to clear a given bit in the INTC\_FRC register to negate a specific level interrupt request. The data value on the register write causes the appropriate bit in the INTC\_FRC register to be cleared. Attempted reads of this register generate an error termination.

This register is provided so interrupt service routines can negate a forced interrupt request without the need to perform a read-modify-write sequence on the INTC\_FRC register.

**Figure 8-7. INTC\_CFRC Register****Table 8-10. INTC\_CFRC Field Descriptions**

Field	Description
7–6	Reserved, must be cleared.
5–0 CLR	For data values within the 56–62 range, the corresponding bit in the INTC_FRC register is cleared, as defined below. 0x38 Bit 56, INTC_FRC[LVL7] is cleared 0x39 Bit 57, INTC_FRC[LVL6] is cleared 0x3A Bit 58, INTC_FRC[LVL5] is cleared 0x3B Bit 59, INTC_FRC[LVL4] is cleared 0x3C Bit 60, INTC_FRC[LVL3] is cleared 0x3D Bit 61, INTC_FRC[LVL2] is cleared 0x3E Bit 62, INTC_FRC[LVL1] is cleared <b>Note:</b> Data values outside this range do not affect the INTC_FRC register. It is recommended the data values be restricted to the 0x38–0x3E (56–62) range to ensure compatibility with future devices.

### 8.3.7 INTC Software and Level-*n* IACK Registers (*n* = 1,2,3,...,7)

The eight read-only interrupt acknowledge (IACK) registers can be explicitly addressed by the memory-mapped accesses or implicitly addressed by a processor-generated interrupt acknowledge cycle during exception processing when CPUCR[IAE] is set. In either case, the interrupt controller's actions are similar.

First, consider an IACK cycle to a specific level, a level-*n* IACK. When this type of IACK arrives in the interrupt controller, the controller examines all 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.

If there is no active interrupt source at the time of the level-*n* IACK, a special spurious interrupt vector (vector number 24 (0x18)) is returned. It is the responsibility of the service routine to manage this error situation.

This protocol implies the interrupting peripheral is not accessed during the acknowledge cycle because the interrupt controller completely services the acknowledge. This means the interrupt source must be explicitly disabled in the peripheral device by the interrupt service routine. This approach provides unique vector capability for all interrupt requests, regardless of the complexity of the peripheral device.

Second, the interrupt controller also supports the concept of a software IACK. This is the ability to query the interrupt controller near the end of an interrupt service routine (after the current interrupt request has been negated) to determine if there are any pending (but currently masked) interrupt requests. If the response to the software IACK's byte operand read is non-zero, the service routine uses the returned value as the vector number of the highest pending interrupt request and passes control to the appropriate new handler. If the returned value is zero, there is no pending interrupt request.

This process avoids the overhead of a context restore and RTE instruction execution, followed immediately by another interrupt exception and context save. In system environments with high rates of interrupt activity, this mechanism can noticeably improve overall performance. For additional details on software IACKs, see [Section 8.6.3, “More on Software IACKs.”](#)

								Access: Read-only
								Offset: CF1_INTC_BASE + 0x20 (INTC_SWIACK) CF1_INTC_BASE + 0x20 + (4× <i>n</i> ) (INTC_LVL <i>n</i> IACK)
								VECN
R	0							
W								
SWIACK Reset	0	0	0	0	0	0	0	0
LVL <i>n</i> IACK Reset	0	0	0	1	1	0	0	0

**Table 8-11. Software and Level-*n* IACK Registers (INTC\_SWIACK, INTC\_LVL*n*IACK)**

**Table 8-12. INTC\_SWIACK, INTC\_LVLnIACK Field Descriptions**

Field	Description
7	Reserved, must be cleared.
6–0 VECN	<p>Vector number. Indicates the appropriate vector number.</p> <p>For the SWIACK register, it is the highest-level, highest-priority request currently being asserted in the CF1_INTC module. If there are no pending requests, VECN is zero.</p> <p>For the LVL<math>n</math>IACK register, it is the highest priority request within the specified level-<math>n</math>. If there are no pending requests within the level, VECN is 0x18 (24) to signal a spurious interrupt.</p>

## 8.4 Functional Description

The basic operation of the CF1\_INTC is detailed in the preceding sections. This section describes special rules applicable to non-maskable level seven interrupt requests and the module's interfaces.

### 8.4.1 Handling of Non-Maskable Level 7 Interrupt Requests

The CPU treats level seven interrupts as non-maskable, edge-sensitive requests, while levels one through six are maskable, level-sensitive requests. As a result of this definition, level seven interrupt requests are a special case. The edge-sensitive nature of these requests means the encoded 3-bit level input from the CF1\_INTC to the V1 ColdFire core must change state before the CPU detects an interrupt. A non-maskable interrupt (NMI) is generated each time the encoded interrupt level changes to level seven (regardless of the SR[I] field) and each time the SR[I] mask changes from seven to a lower value while the encoded request level remains at seven.

## 8.5 Initialization Information

The reset state of the CF1\_INTC module enables the default IRQ mappings and clears any software-forced interrupt requests (INTC\_FRC is cleared). Immediately after reset, the CF1\_INTC begins its cycle-by-cycle evaluation of any asserted interrupt requests and forms the appropriate encoded interrupt level and vector information for the V1 Coldfire processor core. The ability to mask individual interrupt requests using the interrupt controller's IMR is always available, regardless of the level of a particular interrupt request.

## 8.6 Application Information

This section discusses three application topics: emulation of the HCS08's one level interrupt nesting structure, elevating the priority of two IRQs, and more details on the operation of the software interrupt acknowledge (SWIACK) mechanism.

### 8.6.1 Emulation of the HCS08's 1-Level IRQ Handling

As noted in [Table 8-1](#), the HCS08 architecture specifies a 1-level IRQ nesting capability. Interrupt masking is controlled by CCR[I], the interrupt mask flag: clearing CCR[I] enables interrupts, while setting CCR[I]

disables interrupts. The ColdFire architecture defines seven interrupt levels, controlled by the 3-bit interrupt priority mask field in the status register, SR[I], and the hardware automatically supports nesting of interrupts.

To emulate the HCS08's 1-level IRQ capabilities on V1 ColdFire, only two SR[I] settings are used:

- Writing 0 to SR[I] enables interrupts.
- Writing 7 to SR[I] disables interrupts.

The ColdFire core treats the level seven requests as non-maskable, edge-sensitive interrupts.

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 as the first instruction in the ISR. In addition, the V1 instruction set architecture (ISA\_C) includes an 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 that services multiple interrupt requests with different interrupt levels. For more details see the *ColdFire Family Programmer's Reference Manual*. A MOVE-to-SR instruction also performs a similar function.

To emulate the HCS08's 1-level IRQ nesting mechanisms, the ColdFire implementation enables interrupts by clearing SR[I] (typically when using RTE to return to a process) and disables interrupts upon entering every interrupt service routine by one of three methods:

1. Execution of STLDSR #0x2700 as the first instruction of an ISR.
2. Execution of MOVE.w #0x2700,SR as the first instruction of an ISR.
3. Static assertion of CPUCR[IME] that forces the processor to load SR[I] with seven automatically upon the occurrence of an interrupt exception. Because this method removes the need to execute multi-cycle instructions of #1 or #2, this approach improves system performance.

## 8.6.2 Using INTC\_PL6P{7,6} Registers

Section 8.3.3, “INTC Programmable Level 6, Priority {7,6} Registers (INTC\_PL6P{7,6}),” describes control registers that provide the ability to dynamically alter the request level and priority of two IRQs. Specifically, these registers provide the ability to reassign two IRQs to be the highest level 6 (maskable) requests. Consider the following example.

Suppose the system operation desires to remap the receive and transmit interrupt requests of a serial communication device (SCI1) as the highest two maskable interrupts. The default assignments for the SCI1 transmit and receive interrupts are:

- sci1\_rx = interrupt source 16 = vector 80 = level 4, priority 6
- sci1\_tx = interrupt source 17 = vector 81 = level 4, priority 5

To remap these two requests, the INTC\_PL6P{7,6} registers are programmed with the desired interrupt source number:

- Setting INTC\_PL6P7 to 16 (0x10), remaps sci1\_rx as level 6, priority 7.
- Setting INTC\_PL6P6 to 17 (0x11), remaps sci1\_tx as level 6, priority 6.

The reset state of the INTC\_PL6P{7,6} registers disables any request remapping.

### 8.6.3 More on Software IACKs

As previously mentioned, the notion of a software IACK refers to the ability to query the interrupt controller near the end of an interrupt service routine (after the current interrupt request has been cleared) to determine if there are any pending (but currently masked) interrupt requests. If the response to the software IACK's byte operand read is non-zero, the service routine uses the value as the vector number of the highest pending interrupt request and passes control to the appropriate new handler. This process avoids the overhead of a context restore and RTE instruction execution, followed immediately by another interrupt exception and context save. In system environments with high rates of interrupt activity, this mechanism can improve overall system performance noticeably.

To illustrate this concept, consider the following ISR code snippet shown in [Figure 8-8](#).

```

        align    4
        irqxx_entry:
00588: 4fef fff0 lea      -16(sp),sp          # allocate stack space
0058c: 48d7 0303 movem.l #0x0303,(sp)       # save d0/d1/a0/a1 on stack

        irqxx_alternate_entry:
00590:
        ...
        irqxx_swiack:
005c0: 71b8 ffe0 mvz.b   INTC_SWIACK.w,d0    # perform software IACK
005c4: 0c00 0041 cmpi.b  #0x41,d0          # pending IRQ or level 7?
005c8: 6f0a         ble.b   irqxx_exit        # no pending IRQ, then exit
005ca: 91c8         sub.l   a0,a0          # clear a0
005cc: 2270 0c00 move.l  0(a0,d0.1*4),a1    # fetch pointer from xcpt table
005d0: 4ee9 0008 jmp     8(a1)           # goto alternate isr entry point

        align    4
        irqxx_exit:
005d4: 4cd7 0303 movem.l (sp),#0x0303       # restore d0/d1/a0/a1
005d8: 4fef 0010 lea      16(sp),sp          # deallocate stack space
005dc: 4e73         rte                  # return from handler

```

**Figure 8-8. ISR Code Snippet with SWIACK**

This snippet includes the prologue and epilogue for an interrupt service routine as well as code needed to perform software IACK.

At the entry point (`irqxx_entry`), there is a two-instruction prologue to allocate space on the supervisor stack to save the four volatile registers (d0, d1, a0, a1) defined in the ColdFire application binary interface. After saving these registers, the ISR continues at the alternate entry point.

The software IACK is performed near the end of the ISR, after the source of the current interrupt request is negated. First, the appropriate memory-mapped byte location in the interrupt controller is read (PC = 0x5C0). The CF1\_INTC module returns the vector number of the highest priority pending request. If no request is pending, zero is returned. The compare instruction is needed to manage a special case involving pending level seven requests. Because the level seven requests are non-maskable, the ISR is interrupted to service one of these requests. To avoid any race conditions, this check ignores the level seven

## **Interrupt Controller (CF1\_INTC)**

vector numbers. The result is the conditional branch (PC = 0x5C8) is taken if there are no pending requests or if the pending request is a level seven.

If there is a pending non-level seven request, execution continues with a three instruction sequence to calculate and then branch to the appropriate alternate ISR entry point. This sequence assumes the exception vector table is based at address 0x(00)00\_0000 and that each ISR uses the same two-instruction prologue shown here. The resulting alternate entry point is a fixed offset (8 bytes) from the normal entry point defined in the exception vector table.

The ISR epilogue includes a three instruction sequence to restore the volatile registers from the stack and return from the interrupt exception.

This example is intentionally simple, but does show how performing the software IACK and passing control to an alternate entry point when there is a pending but masked interrupt request can avoid the execution of the ISR epilogue, another interrupt exception, and the ISR prologue.

# Chapter 9

## Parallel Input/Output Control

### 9.1 Overview of MCF51CN128 I/O Functions

#### 9.1.1 Summary

This section explains software controls related to parallel input/output (I/O) and pin control. The MCF51CN128 series MCUs have up to nine parallel I/O ports which include a total of 70 I/O pins. See [Chapter 2, “Pins and Connections,”](#) for more information about pin assignments and external hardware considerations of these pins. [Table 9-1](#) summarizes capabilities for the MCF51CN128 on a per port basis.

**Table 9-1. Functionality on a Per Port Basis**

Port Name	Width	GPIO Implementation <sup>1</sup>	Keyboard Interface	Pin Control <sup>2</sup>
PTA	8	GPIO[7:0]	—	PE, SE, DS, IFE
PTB	8	GPIO[7:0]	—	PE, SE, DS, IFE
PTC	8	GPIO[7:0]	—	PE, SE, DS, IFE
PTD <sup>3</sup>	8	GPIO/RGPIO[7:0]	—	PE, SE, DS, IFE
PTE	8	GPIO[7:0]	KBI2	PE, SE, DS, IFE
PTF	8	GPIO/RGPIO[15:8]	—	PE, SE, DS, IFE
PTG	8	GPIO[7:0]	KBI1	PE, SE, DS, IFE
PTH	8	GPIO[7:0]	—	PE, SE, DS, IFE
PTJ	6	GPIO[7:0]	—	PE, SE, DS, IFE

<sup>1</sup> D[6] GPIO is output only.

<sup>2</sup> PE = pull-up enable, SE = slew rate enable, DS = drive strength control, IFE = input filter enable

<sup>3</sup> When PTD6 is set as RGPIO output, and "1" is driven to PTD6 by the RGPIO function, a read of register RGPIODATA6 always return a 0 because V1 RGPIO design looks for IO enable when the return value of RGPIO function reads data. As PTD6 is set to RGPIO output only, it returns 0 always to RGPIODATA6, although PTD6 pin is driven high.

#### 9.1.2 Ports Implemented via Rapid GPIO

Ports D and F are implemented using Rapid GPIO functions which are integrated as part of the ColdFire core to improve edge resolution on those pins. RGPIO provides functionality similar to standard GPIO with the addition of set/clear/toggle functionality, but at CPU (rather than peripheral bus clock) rates.

Port D is associated with GPIO[7:0]. [Table 9-2](#) shows GPIO pin mapping to the port I/O pins.

**Table 9-2. Port D Pin Mapping to GPIO**

Port pin	PTD7	PTD6	PTD5	PTD4	PTD3	PTD2	PTD1	PTD0
<b>KBI1 pin</b>	GPIO7	GPIO6	GPIO5	GPIO4	GPIO3	GPIO2	GPIO1	GPIO0

Port F is associated with GPIO[15:8]. [Table 9-2](#) shows GPIO pin mapping to the port I/O pins.

**Table 9-3. Port F Pin Mapping to GPIO**

Port pin	PTF7	PTF6	PTF5	PTF4	PTF3	PTF2	PTF1	PTF0
<b>KBI1 pin</b>	GPIO15	GPIO14	GPIO13	GPIO12	GPIO11	GPIO10	GPIO9	GPIO8

See [Chapter 10, “Rapid GPIO \(GPIO\),”](#) for additional details.

### 9.1.3 Keyboard Functions

Ports E and G include keyboard interrupt capability. Keyboard 1 is associated with GPIO Port G. [Table 9-5](#) shows KBI1 pin mapping to the port I/O pins.

**Table 9-4. KBI1 Pin Mapping**

Port pin	PTG7	PTG6	PTG5	PTG4	PTG3	PTG2	PTG1	PTG0
<b>KBI1 pin</b>	KBI1P7	KBI1P6	KBI1P5	KBI1P4	KBI1P3	KBI1P2	KBI1P1	KBI1P0

Keyboard 2 is associated with GPIO Port E. [Table 9-5](#) shows KBI2 pin mapping to the port I/O pins.

**Table 9-5. KBI2 Pin Mapping**

Port pin	PTE7	PTE6	PTE5	PTE4	PTE3	PTE2	PTE1	PTE0
<b>KBI2 pin</b>	KBI2P7	KBI2P6	KBI2P5	KBI2P4	KBI2P3	KBI2P2	KBI2P1	KBI2P0

### 9.1.4 Port Mux Control

Many port pins are shared with on-chip peripherals such as timer systems, communication systems, or keyboard interrupts as shown in [Figure 1-1](#). You may select which function owns a pin with the port mux control (MC) registers. These are defined in detail in [Section 9.7](#).

### 9.1.5 Special Cases

#### 9.1.5.1 Pull-Up Resistors

After reset, the shared peripheral functions are normally disabled and the pins are configured as inputs ( $PTxDD[n] = 0$ ). The pin control functions for each pin are configured as follows:

- Slew rate control enabled ( $PTxSE[n] = 1$ )

- Low drive strength selected ( $\text{PTxDS}[n] = 0$ )
- Internal pull-ups disabled ( $\text{PTxPE}[n] = 0$ )

Two exceptions to this are the  $\overline{\text{RESET}}$  and BKGD/MS pins, which have pull-ups enabled at reset.

### 9.1.5.2 Port J

Port J has only 6 pins, versus the usual 8, associated with it. Always write the upper two bits their reset defaults described in the programming models sections.

### 9.1.5.3 $\overline{\text{RESET}}$

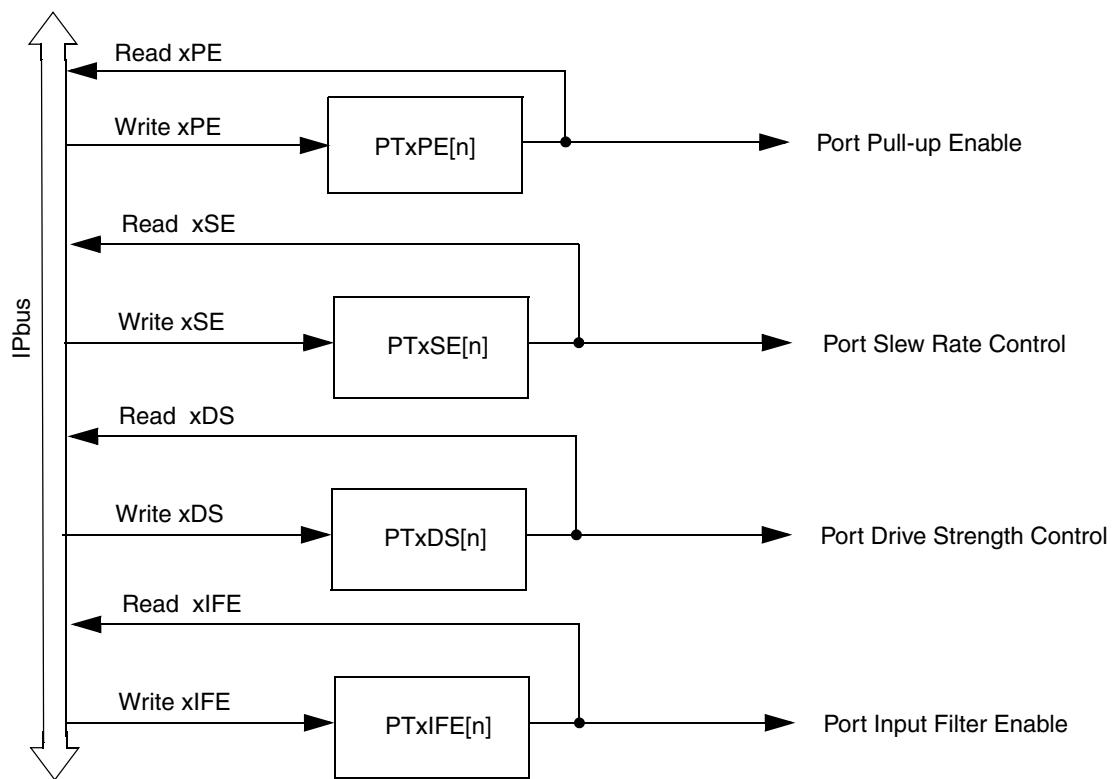
$\overline{\text{RESET}}$  is an open drain signal, and cannot be programmed to an active high.

## 9.2 Pin Controls

This section shows the superset of pin control functions which may be present on V1 ColdFire devices. Some devices may not include all of these controls. See the summary table earlier in the chapter for specific capabilities for your device. See [Section 2.3, “Pin Mux Controls”](#) for information on Pin Mux Control registers.

### 9.2.1 Pin Controls Overview

A set of registers (shown in [Figure 9-1](#)) control pull-ups, slew rate, drive strength and input filter enables for the pins. They may also be used in conjunction with the peripheral functions on these pins. These registers are associated with the parallel I/O ports and Rapid GPIO (RGPIO) ports, but operate independently of both.



**Figure 9-1. Pin Control Logic Block Diagram**

### 9.2.2 Pin Controls Programming Model

These registers control the pull-ups, slew rate, drive strength, and input filter for all the pins and may be used for the peripheral functions (FEC, Mini-FlexBus, etc.) on these pins.

**NOTE**

The full complement of pin controls may not be present on all Freescale devices. See the summary table earlier in the chapter to determine which of these are present on your device.

**Table 9-6. Register Set Summary**

Register	Description	Access
PTxPE	Port x Pull Enable Register	read/write
PTxSE	Port x Slew Rate Enable Register	read/write
PTxDS	Port x Drive Strength Selection Register	read/write
PTxIFE	Port x Input Filter Enable Register	read/write

Refer to tables in [Chapter 4, “Memory,”](#) for the absolute address assignments for all registers. This section refers to registers and control bits only by their names.

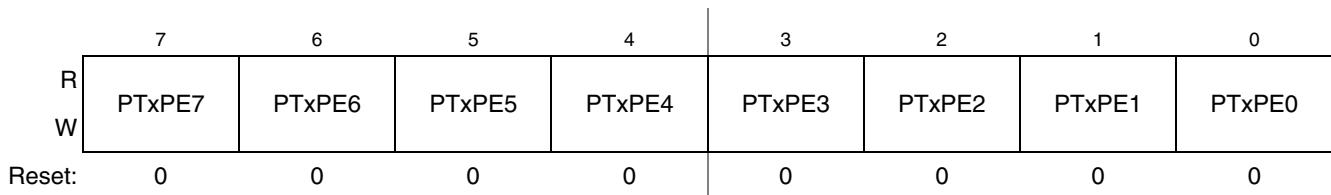
**NOTE**

A Freescale Semiconductor-provided equate or header file normally is used to translate these names into the appropriate absolute addresses.

**9.2.2.1 Port x Pull Enable Register (PTxPE)**

An internal pull-up device can be enabled for each port pin by setting the corresponding bit in the pull-up enable register (PTxPE[n]). The pull-up device is disabled if the pin is either:

- Configured as an output by the parallel I/O control logic
- Configured as a shared peripheral function
- Controlled by an analog function.
- At reset, except for RESETB and BKGD/MS.

**Figure 9-2. Internal Pull Enable for Port x Register (PTxPE)****Table 9-7. PTxPE Field Descriptions**

Field	Description
7-0 PTxPE <sub>n</sub>	<b>Internal Pull Enable for Port x Bits</b> — Each of these control bits determines if the internal pull-up device is enabled for the associated PTx pin. For Port x pins that are configured as outputs, these bits have no effect and the internal pull devices are disabled. 0 Internal pull-up device disabled for Port x bit <i>n</i> . 1 Internal pull-up device enabled for Port x bit <i>n</i> .

### 9.2.2.2 Port x Slew Rate Enable Register (PTxSE)

Slew rate control can be enabled for each port pin by setting the corresponding bit in the slew rate control register (PTxSE[n]). When enabled, slew control limits the rate at which an output can transition in order to reduce EMC emissions. Slew rate control has no effect on pins that are configured as inputs.

	7	6	5	4	3	2	1	0
R W	PTxSE7	PTxSE6	PTxSE5	PTxSE4	PTxSE3	PTxSE2	PTxSE1	PTxSE0
Reset:	0	0	0	0	0	0	0	0

Figure 9-3. Slew Rate Enable for Port x Register (PTxSE)

Table 9-8. PTxSE Field Descriptions

Field	Description
7–0 PTxSE $n$	<b>Output Slew Rate Enable for Port x Bits</b> — Each of these control bits determines if the output slew rate control is enabled for the associated PTx pin. For Port x pins configured as inputs, these bits have no effect. 0 Output slew rate control disabled for Port x bit $n$ . 1 Output slew rate control enabled for Port x bit $n$ .

### 9.2.2.3 Port x Drive Strength Selection Register (PTxDS)

An output pin can be selected to have high output drive strength by setting the corresponding bit in the drive strength select register (PTxDS[n]). When high drive is selected, a pin is capable of sourcing and sinking greater current. Even though every I/O pin can be selected as high drive, ensure that the total current source and sink limits for the MCU are not exceeded. Drive strength selection is intended to affect the DC behavior of I/O pins. However, the AC behavior is also affected. High drive allows a pin to drive a greater load with the same switching speed as a low drive enabled pin into a smaller load. Because of this, the EMC emissions may be affected by enabling pins as high drive.

	7	6	5	4	3	2	1	0
R W	PTxDS7	PTxDS6 <sup>1</sup>	PTxDS5	PTxDS4	PTxDS3	PTxDS2	PTxDS1	PTxDS0
Reset:	0	1	0	0	0	0	0	0

Figure 9-4. Drive Strength Selection for Port x Register (PTxDS)

<sup>1</sup> The PTD[6](BKGD/MS) pin's drive strength can not be changed by this register and is always high.

Table 9-9. PTxDS Field Descriptions

Field	Description
7–0 PTxDS $n$	<b>Output Drive Strength Selection for Port x Bits</b> — Each of these control bits selects between low and high output drive for the associated PTx pin. For Port x pins configured as inputs, these bits have no effect. 0 Low output drive strength selected for Port x bit $n$ . 1 High output drive strength selected for Port x bit $n$ .

### 9.2.2.4 Port x Input Filter Enable Register (PTxIFE)

The pad cells on this device incorporate optional low pass filters on the digital input functions. These are enabled by setting the appropriate bit in the input filter enable register (PTxIFE[n]). When set, a low pass filter (10MHz to 30MHz bandwidth) is enabled in the logic input path. When cleared, the filter is bypassed.

	7	6	5	4	3	2	1	0
R W	PTxIFE7	PTxIFE6	PTxIFE5	PTxIFE4	PTxIFE3	PTxIFE2	PTxIFE1	PTxIFE0
Reset:	0	0	0	0	0	0	0	0

Figure 9-5. Port x Input Filter Enable Register (PTxIFE)

Table 9-10. PTxIFE Field Descriptions

Field	Description
7–0 PTxIFEn	<b>Port x Input Filter Enable</b> — Input low-pass filter enable control bits for PTx pins. 0 Input filter disabled 1 Input filter enabled

## 9.3 Standard GPIO

### 9.3.1 GPIO Overview

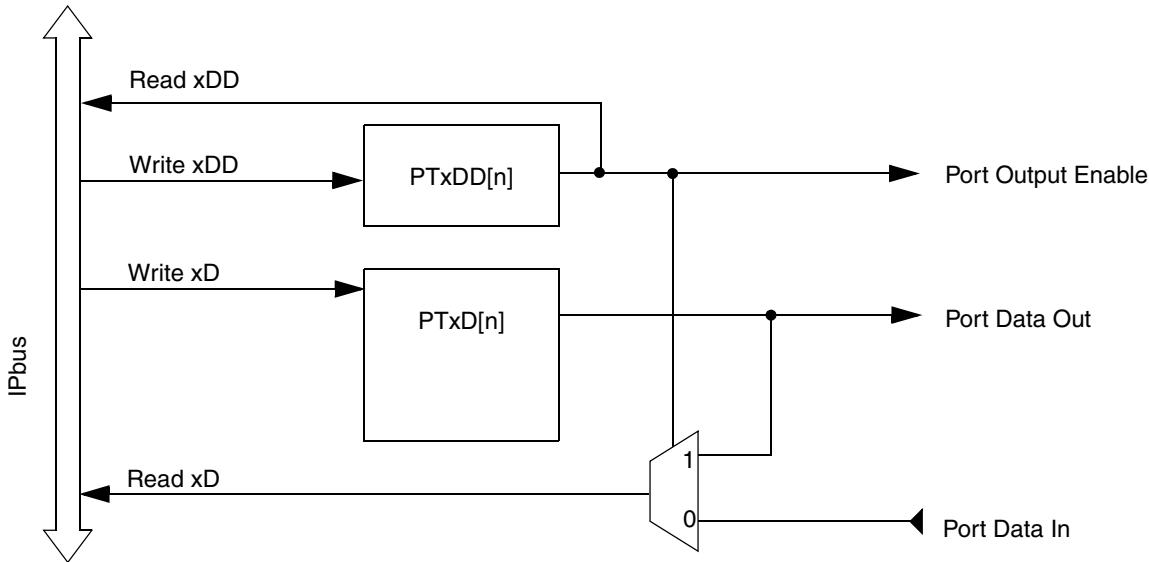
Reading and writing of parallel I/Os are performed through the port data registers. The direction, either input or output, is controlled through the port data direction registers. The parallel I/O port function for an individual pin is illustrated in the block diagram shown in [Figure 9-6](#).

The data direction control bit (PTxDD[n]) determines whether the output buffer for the associated pin is enabled, and also controls the source for port data register reads. The input buffer for the associated pin is always enabled unless the pin is enabled as an analog function or is an output-only pin.

When a shared digital function is enabled for a pin, the output buffer is controlled by the shared function. However, the data direction register bit continues to control the source for reads of the port data register.

When a shared analog function is enabled for a pin, the input and output buffers are disabled. A value of 0 is read for any port data bit where the bit is an input (PTxDD[n] = 0) and the input buffer is disabled.

Writing to the port data register before changing the direction of a port pin to an output ensures that the pin is not driven momentarily with an old data value in the port data register.

**Figure 9-6. GPIO Bit Block Diagram**

Pin mux controls leave GPIO input buffers enabled for all digital functions, regardless of mux control selection. This gives you the ability to poll a port to decode a key pressed after a keyboard interrupt is fired. The same function can be used for IRQ pin level detection and debounce software.

**NOTE**

- Use pin mux control registers from [Section 2.3, “Pin Mux Controls”](#) to assign GPIO signals to the MCF51CN128 package pins.
- Most pin functions default to GPIO and must be software configured.

### 9.3.2 GPIO Programming Model

**Table 9-11. Register Set Summary**

Register	Description	Access
PTxD	PORTx Data Register	read/write
PTxDD	PORTx Data Direction Register	read/write

Refer to tables in [Chapter 4, “Memory,”](#) for the absolute address assignments for all registers. This section refers to registers and control bits only by their names.

**NOTE**

A Freescale Semiconductor-provided equate or header file normally is used to translate these names into the appropriate absolute addresses.

### 9.3.2.1 Port x Data Register (PTxD)

The data register of each port allows software to interact with the pins of the chip. Each bit of each data register controls one pin on the chip. When the port bit is configured as an input (PTxDD[n]=0), a read of the port returns the logic value of the external pin input for that bit location. When the port bit is configured as an output (PTxDD[n]=1), a read of the port returns the state of the internal data register bit.

Writing to the data register changes the values of all internal data register bits to the value being written regardless of the associated DD bit.

	7	6	5	4	3	2	1	0
R W	PTxD7	PTxD6	PTxD5	PTxD4	PTxD3	PTxD2	PTxD1	PTxD0
Reset:	0	0	0	0	0	0	0	0

Figure 9-7. Port x Data Register (PTxD)

Table 9-12. PTxD Field Descriptions

Field	Description
7–0 PTxD $n$	<b>Port x Data Register Bits</b> — For Port x pins that are inputs, reads return the logic level on the pin. For Port x pins that are configured as outputs, reads return the last value written to this register. Writes are latched into all bits of this register. For Port x pins that are configured as outputs, the logic level is driven out the corresponding MCU pin. Reset forces PTxD to all zeroes. Since reset configures all port pins as high-impedance inputs, these zeroes are not driven onto the pins.

### 9.3.2.2 Port x Data Direction Register (PTxDD)

The PTxDD registers control the direction of the port x pins.

	7	6	5	4	3	2	1	0
R W	PTxDD7	PTxDD6	PTxDD5	PTxDD4	PTxDD3	PTxDD2	PTxDD1	PTxDD0
Reset:	0	0	0	0	0	0	0	0

Figure 9-8. Port x Data Direction Register (PTxDD)

Table 9-13. PTxDD Field Descriptions

Field	Description
7–0 PTxDD $n$	<b>Data Direction for Port x Bits</b> — These read/write bits control the direction of Port x pins and what is read for PTxD reads. 0 Input (output driver disabled) and PTxD reads return the pin value. 1 Output driver enabled for Port x bit $n$ and PTxD reads return the contents of PTxD $n$ .

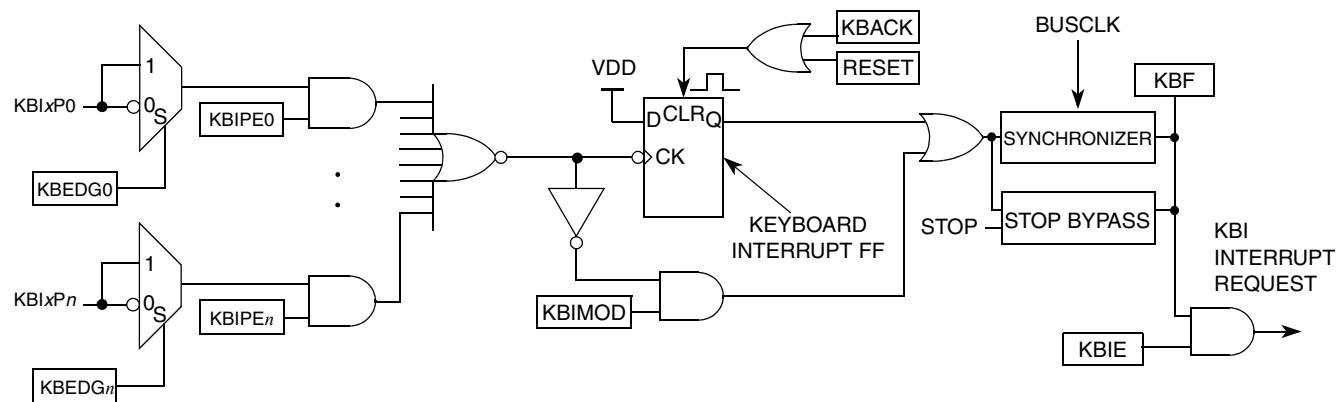
## 9.4 V1 ColdFire Rapid GPIO Functionality

The V1 ColdFire core can perform higher speed I/O via its local bus, which does not have latency penalties associated with the on-chip peripheral bus bridge. The Rapid GPIO module contains separate set/clear/data registers based at address 0x(00)C0\_0000. This functionality can be programmed to take priority on ports A and B.

This functionality is further defined in [Chapter 10, “Rapid GPIO \(RGPIO\).”](#)

## 9.5 Keyboard Interrupts

The block diagram for each keyboard interrupt logic is shown [Figure 9-9](#).



**Figure 9-9. Port Interrupt Block Diagram**

Writing to KBIxPE[KBIPEn] independently enables or disables each port pin. Each port can be configured as edge-sensitive or edge- and level-sensitive based on the KBIxSC[KBIMOD] bit. Edge sensitivity can be software programmed to be falling or rising; the level can be either low or high. The polarity of the edge-sensitivity or edge- and level-sensitivity is selected using the KBIxES[KBEDGn].

Synchronous logic is used to detect edges. Prior to detecting an edge, enabled port inputs must be at the deasserted logic level. A falling edge is detected when an enabled port input signal is seen as a logic 1 (the deasserted level) during one bus cycle and then a logic 0 (the asserted level) during the next cycle. A rising edge is detected when the input signal is seen as a logic 0 during one bus cycle and then a logic 1 during the next cycle.

### 9.5.1 Keyboard Functional Considerations

#### 9.5.1.1 Edge Only Sensitivity

A valid edge on an enabled port pin sets KBIxSC[KBF]. If KBIxSC[KBIE] is set, an interrupt request is generated to the CPU. Write a 1 to KBIxSC[KBACK] to clear KBF.

### 9.5.1.2 Edge and Level Sensitivity

A valid edge or level on an enabled port pin sets the KBIxSC[KBF] bit. If KBIxSC[KBIE] is set, an interrupt request is generated to the CPU. Write a 1 to KBIxSC[KBACK] to clear KBF, provided all enabled port inputs are at their deasserted levels. KBF remains set if any enabled port pin is asserted while attempting to clear by writing a 1 to KBACK.

### 9.5.1.3 Pull-up/Pull-down Resistors

The keyboard interrupt pins can be configured to use an internal pull-up/pull-down resistor using the associated I/O port pull-up enable register. If an internal resistor is enabled, the KBIxES register is used to select whether the resistor is a pull-up (KBEDG[n] = 0) or a pull-down (KBEDG[n] = 1).

### 9.5.1.4 Keyboard Interrupt Initialization

When an interrupt pin is first enabled, it is possible to get a false interrupt flag. To prevent a false interrupt request during pin interrupt initialization, do the following:

1. Mask interrupts by clearing KBIxSC[KBIE].
2. Select the pin polarity by setting the appropriate KBIxES[n] bits.
3. If using internal pull-up/pull-down device, configure the associated pull enable bits in PTxPE.
4. Enable the interrupt pins by setting the appropriate KBIxPE[n] bits.
5. Write 1 to KBIxSC[KBACK] to clear any false interrupts.
6. Set KBIxSC[KBIE] to enable interrupts.

## 9.5.2 Keyboard Programming Model

**Table 9-14. Register Set Summary**

Register	Description	Access
KBIxSC	Keyboard Interrupt Status & Control Register	read/write
KBIxPE	Keyboard Interrupt Pin Select Register	read/write
KBIxES	KBIx Interrupt Edge Select Register	read/write

Refer to tables in [Chapter 4, “Memory,”](#) for the absolute address assignments for all registers. This section refers to registers and control bits only by their names.

#### NOTE

A Freescale Semiconductor-provided equate or header file normally is used to translate these names into the appropriate absolute addresses.

### 9.5.2.1 KBIx Interrupt Status and Control Register (KBIxSC)

	7	6	5	4		3	2	1	0
R	0	0	0	0	KBF	0		KBIE	KBIMOD
W						KBACK		0	0
Reset:	0	0	0	0		0	0	0	0

Figure 9-10. KBIx Interrupt Status and Control Register (KBIxSC)

Table 9-15. KBIxSC Field Descriptions

Field	Description
7–4	Reserved, must be cleared.
3 KBF	<b>KBIx Interrupt Flag</b> — KBF indicates when a KBIx interrupt is detected. Writes have no effect on KBF. 0 No KBIx interrupt detected. 1 KBIx interrupt detected.
2 KBACK	<b>KBIx Interrupt Acknowledge</b> — Writing a 1 to KBACK is part of the flag clearing mechanism. KBACK always reads as 0.
1 KBIE	<b>KBIx Interrupt Enable</b> — KBIE determines whether a KBIx interrupt is requested. 0 KBIx interrupt request not enabled. 1 KBIx interrupt request enabled.
0 KBIMOD	<b>KBIx Detection Mode</b> — KBIMOD (along with the KBIxES bits) controls the detection mode of the KBIx interrupt pins. 0 KBIx pins detect edges only. 1 KBIx pins detect edges and levels.

### 9.5.2.2 KBIx Interrupt Pin Select Register (KBIxPE)

	7	6	5	4		3	2	1	0
R	KBIPE7	KBIPE6	KBIPE5	KBIPE4	KBIPE3	KBIPE2	KBIPE1	KBIPE0	
W									
Reset:	0	0	0	0		0	0	0	0

Figure 9-11. KBIx Interrupt Pin Select Register (KBIxPE)

Table 9-16. KBIxPE Field Descriptions

Field	Description
7–0 KBIPE $n$	<b>KBIx Interrupt Pin Selects</b> — Each of the KBIPE $n$ bits enable the corresponding KBIx interrupt pin. 0 Pin not enabled as interrupt. 1 Pin enabled as interrupt.

### 9.5.2.3 KBIx Interrupt Edge Select Register (KBIxES)

	7	6	5	4	3	2	1	0
R W	KBEDG7	KBEDG6	KBEDG5	KBEDG4	KBEDG3	KBEDG2	KBEDG1	KBEDG0
Reset:	0	0	0	0	0	0	0	0

Figure 9-12. KBIx Edge Select Register (KBIxES)

Table 9-17. KBIxES Field Descriptions

Field	Description
7–0 KBEDGn	<b>KBIx Edge Selects</b> — Each of the KBEDG $n$ bits serves a dual purpose by selecting the polarity of the active interrupt edge as well as selecting a pull-up or pull-down device if enabled. 0 A pull-up device is connected to the associated pin and detects falling edge/low level for interrupt generation. 1 A pull-down device is connected to the associated pin and detects rising edge/high level for interrupt generation.

## 9.6 Pin Behavior in Stop Modes

Pin behavior following execution of a STOP instruction depends on the stop mode that is entered. An explanation of pin behavior for the various stop modes follows:

- Stop2 mode is a partial power-down mode, whereby I/O latches are maintained in their state as before the STOP instruction was executed (port states are lost and need to be restored upon exiting stop2). CPU register status and the state of I/O registers should be saved in RAM before the executing the STOP instruction to place the MCU in Stop2 mode.  
After recovery from Stop2 mode, before accessing any I/O, examine the state of the SPMSC2[PPDF] bit.
  - If the PPDF bit is cleared, I/O must be initialized as if a power-on-reset had occurred.
  - If the PPDF bit is set, I/O register states should be restored from the values saved in RAM before the STOP instruction was executed and peripherals may require initialization or restoration to their pre-stop condition.
 Then write a 1 to the SPMSC2[PPDACK] bit. Access to I/O is now permitted again in the user application program.
- In Stop3 and Stop4 modes, all I/O is maintained because internal logic circuitry stays powered. After recovery, normal I/O function is available to the user.

# Chapter 10

## Rapid GPIO (RGPIO)

### 10.1 Introduction

The Rapid GPIO (RGPIO) module provides a 16-bit general-purpose I/O module directly connected to the processor's high-speed 32-bit local bus. This connection plus support for single-cycle, zero wait-state data transfers allows the RGPIO module to provide improved pin performance when compared to more traditional GPIO modules located on the internal slave peripheral bus.

Many of the pins associated with a device 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 (GPIO) pins. The definition of the exact pin functions and the affected signals is specific to each device. Every GPIO port, including the RGPIO module, has registers that configure, monitor, and control the port pins.

#### NOTE

- Use pin mux control registers from [Section 2.3, “Pin Mux Controls”](#) to assign RGPIO signals to the MCF51CN128 package pins.
- Most pin functions default to GPIO and must be software configured before using RGPIO.

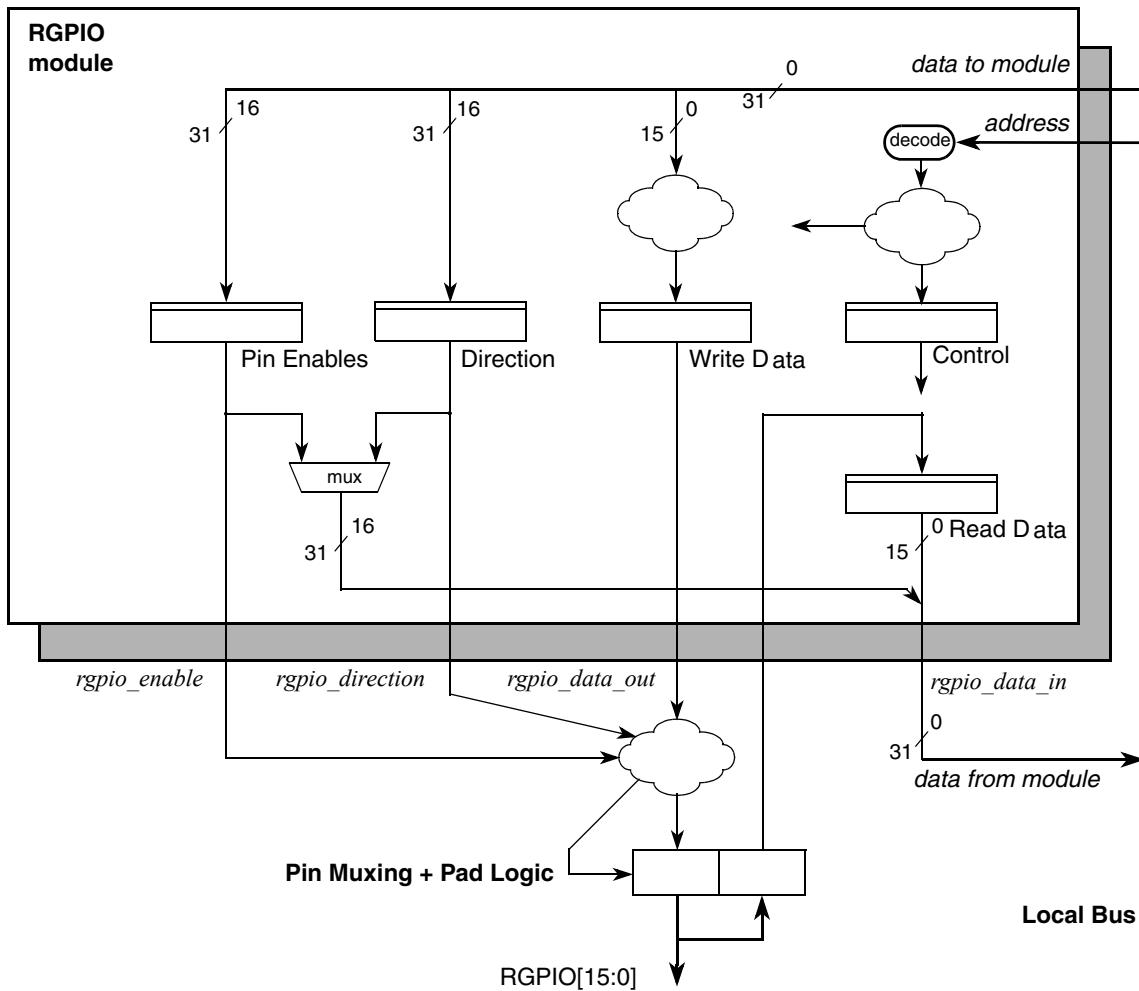
#### 10.1.1 Overview

The RGPIO module provides 16-bits of high-speed GPIO functionality, mapped to the processor's bus. The key features of this module include:

- 16 bits of high-speed GPIO functionality connected to the processor's local 32-bit bus
- Memory-mapped device connected to the ColdFire core's local bus
  - Support for all access sizes: byte, word, and longword
  - All reads and writes complete in a single data phase cycle for zero wait-state response
- Data bits can be accessed directly or via alternate addresses to provide set, clear, and toggle functions
  - Alternate addresses allow set, clear, toggle functions using simple store operations without the need for read-modify-write references
- Unique data direction and pin enable control registers
- Package pin toggle rates typically 1.5–3.5x faster than comparable pin mapped onto peripheral bus

## Rapid GPIO (RGPIO)

A simplified block diagram of the RGPIO module is shown in [Figure 10-1](#). The details of the pin muxing and pad logic are device-specific.



**Figure 10-1. RGPIO Block Diagram**

### 10.1.2 Features

The major features of the RGPIO module providing 16 bits of high-speed general-purpose input/output are:

- Small memory-mapped device connected to the processor's local bus
  - All memory references complete in a single cycle to provide zero wait-state responses
  - Located in processor's high-speed clock domain
- Simple programming model
  - Four 16-bit registers, mapped as three program-visible locations
    - Register for pin enables
    - Register for controlling the pin data direction
    - Register for storing output pin data

- Register for reading current pin state
- The two data registers (read, write) are mapped to a single program-visible location
- Alternate addresses to perform data set, clear, and toggle functions using simple writes
- Separate read and write programming model views enable simplified driver software
- Support for any access size (byte, word, or longword)

### 10.1.3 Modes of Operation

The RGPIO module does not support any special modes of operation. As a memory-mapped device located on the processor's high-speed local bus, it responds based strictly on memory address and does not consider the operating mode (supervisor, user) of its references.

## 10.2 External Signal Description

### 10.2.1 Overview

As shown in [Figure 10-1](#), the RGPIO module's interface to external logic is indirect via the device pin-muxing and pad logic. For a list of the associated RGPIO input/output signals, see [Table 10-1](#).

**Table 10-1. RGPIO Module External I/O Signals**

Signal Name	Type	Description
GPIO[15:0]	I/O	RGPIO Data Input/Output

### 10.2.2 Detailed Signal Descriptions

[Table 10-2](#) provides descriptions of the RGPIO module's input and output signals.

**Table 10-2. RGPIO Detailed Signal Descriptions**

Signal	I/O	Description				
GPIO[15:0]	I/O	<p>Data Input/Output. When configured as an input, the state of this signal is reflected in the read data register. When configured as an output, this signal is the output of the write data register.</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 15%; padding: 5px;"><b>State Meaning</b></td> <td style="width: 85%; padding: 5px;">           Asserted—            Input: Indicates the GPIO pin was sampled as a logic high at the time of the read.            Output: Indicates a properly-enabled GPIO output pin is to be driven high.            Negated—            Input: Indicates the GPIO pin was sampled as a logic low at the time of the read.            Output: Indicates a properly-enabled GPIO output pin is to be driven low.         </td> </tr> <tr> <td style="padding: 5px;"><b>Timing</b></td> <td style="padding: 5px;">           Assertion/Negation—            Input: Anytime. The input signal is sampled at the rising-edge of the processor's high-speed clock on the data phase cycle of a read transfer of this register.            Output: Occurs at the rising-edge of the processor's high-speed clock on the data phase cycle of a write transfer to this register. This output is asynchronously cleared by system reset.         </td> </tr> </table>	<b>State Meaning</b>	Asserted— Input: Indicates the GPIO pin was sampled as a logic high at the time of the read. Output: Indicates a properly-enabled GPIO output pin is to be driven high. Negated— Input: Indicates the GPIO pin was sampled as a logic low at the time of the read. Output: Indicates a properly-enabled GPIO output pin is to be driven low.	<b>Timing</b>	Assertion/Negation— Input: Anytime. The input signal is sampled at the rising-edge of the processor's high-speed clock on the data phase cycle of a read transfer of this register. Output: Occurs at the rising-edge of the processor's high-speed clock on the data phase cycle of a write transfer to this register. This output is asynchronously cleared by system reset.
<b>State Meaning</b>	Asserted— Input: Indicates the GPIO pin was sampled as a logic high at the time of the read. Output: Indicates a properly-enabled GPIO output pin is to be driven high. Negated— Input: Indicates the GPIO pin was sampled as a logic low at the time of the read. Output: Indicates a properly-enabled GPIO output pin is to be driven low.					
<b>Timing</b>	Assertion/Negation— Input: Anytime. The input signal is sampled at the rising-edge of the processor's high-speed clock on the data phase cycle of a read transfer of this register. Output: Occurs at the rising-edge of the processor's high-speed clock on the data phase cycle of a write transfer to this register. This output is asynchronously cleared by system reset.					

## 10.3 Memory Map/Register Definition

The RGPIO module provides a compact 16-byte programming model based at a system memory address of 0x(00)C0\_0000 (noted as RGPIO\_BASE throughout the chapter). As previously noted, the programming model views are different between reads and writes as this enables simplified software for manipulation of the RGPIO pins.

Additionally, the RGPIO programming model is defined with a 32-bit organization. The basic size of each program-visible register is 16 bits, but the programming model may be referenced using byte (8-bit), word (16-bit) or longword (32-bit) accesses. Performance is typically maximized using 32-bit accesses.

### NOTE

Writes to the two-byte fields at RGPIO\_BASE + 0x8 and RGPIO\_BASE + 0xC are allowed, but do not affect any program-visible register within the RGPIO module.

**Table 10-3. RGPIO Write Memory Map**

Offset Address	Register	Width (bits)	Access	Reset Value	Section/Page
0x00	RGPIO Data Direction Register (RGPIO_DIR)	16	W	0x0000	<a href="#">10.3.1/10-5</a>
0x02	RGPIO Write Data Register (RGPIO_DATA)	16	W	0x0000	<a href="#">10.3.2/10-5</a>
0x04	RGPIO Pin Enable Register (RGPIO_ENB)	16	W	0x0000	<a href="#">10.3.3/10-6</a>
0x06	RGPIO Write Data Clear Register (RGPIO_CLR)	16	W	N/A	<a href="#">10.3.4/10-6</a>
0x0A	RGPIO Write Data Set Register (RGPIO_SET)	16	W	N/A	<a href="#">10.3.5/10-7</a>
0x0E	RGPIO Write Data Toggle Register (RGPIO_TOG)	16	W	N/A	<a href="#">10.3.6/10-7</a>

**Table 10-4. RGPIO Read Memory Map**

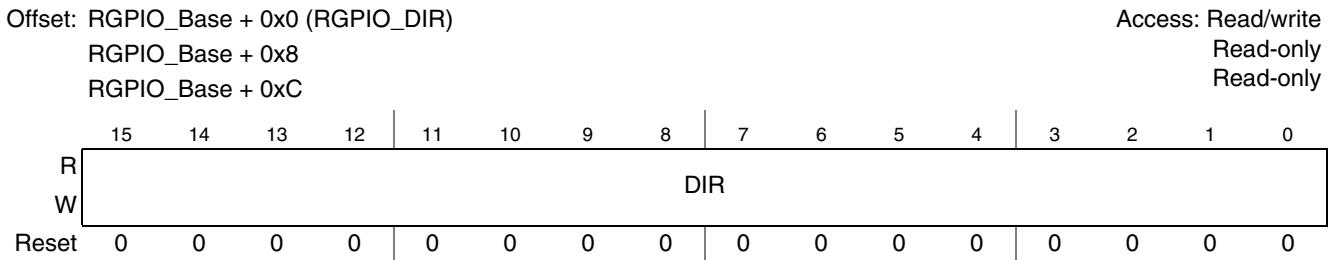
Offset Address	Register	Width (bits)	Access	Reset Value	Section/Page
0x00	RGPIO Data Direction Register (RGPIO_DIR)	16	R	0x0000	<a href="#">10.3.1/10-5</a>
0x02	RGPIO Write Data Register (RGPIO_DATA)	16	R	0x0000	<a href="#">10.3.2/10-5</a>
0x04	RGPIO Pin Enable Register (RGPIO_ENB)	16	R	0x0000	<a href="#">10.3.3/10-6</a>
0x06	RGPIO Write Data Register (RGPIO_DATA)	16	R	0x0000	<a href="#">10.3.2/10-5</a>
0x08	RGPIO Data Direction Register (RGPIO_DIR)	16	R	0x0000	<a href="#">10.3.1/10-5</a>
0x0A	RGPIO Write Data Register (RGPIO_DATA)	16	R	0x0000	<a href="#">10.3.2/10-5</a>
0x0C	RGPIO Data Direction Register (RGPIO_DIR)	16	R	0x0000	<a href="#">10.3.1/10-5</a>
0x0E	RGPIO Write Data Register (RGPIO_DATA)	16	R	0x0000	<a href="#">10.3.2/10-5</a>

### 10.3.1 RGPI0 Data Direction (RGPI0\_DIR)

The read/write RGPI0\_DIR register defines whether a properly-enabled RGPI0 pin is configured as an input or output:

- Setting any bit in RGPI0\_DIR configures a properly-enabled RGPI0 port pin as an output
- Clearing any bit in RGPI0\_DIR configures a properly-enabled RGPI0 port pin as an input

At reset, all bits in the RGPI0\_DIR are cleared.



**Figure 10-2. RGPI0 Data Direction Register (RGPI0\_DIR)**

**Table 10-5. RGPI0\_DIR Field Descriptions**

Field	Description
15–0 DIR	Data direction. 0 A properly-enabled RGPI0 pin is configured as an input 1 A properly-enabled RGPI0 pin is configured as an output

### 10.3.2 RGPI0 Data (RGPI0\_DATA)

The RGPI0\_DATA register specifies the write data for a properly-enabled RGPI0 output pin or the sampled read data value for a properly-enabled input pin. An attempted read of the RGPI0\_DATA register returns undefined data for disabled pins, since the data value is dependent on the device-level pin muxing and pad implementation. The RGPI0\_DATA register is read/write. At reset, all bits in the RGPI0\_DATA registers are cleared.

To set bits in a RGPI0\_DATA register, directly set the RGPI0\_DATA bits or set the corresponding bits in the RGPI0\_SET register. To clear bits in the RGPI0\_DATA register, directly clear the RGPI0\_DATA bits, or clear the corresponding bits in the RGPI0\_CLR register. Setting a bit in the RGPI0\_TOG register inverts (toggles) the state of the corresponding bit in the RGPI0\_DATA register.

## Rapid GPIO (RGPIO)

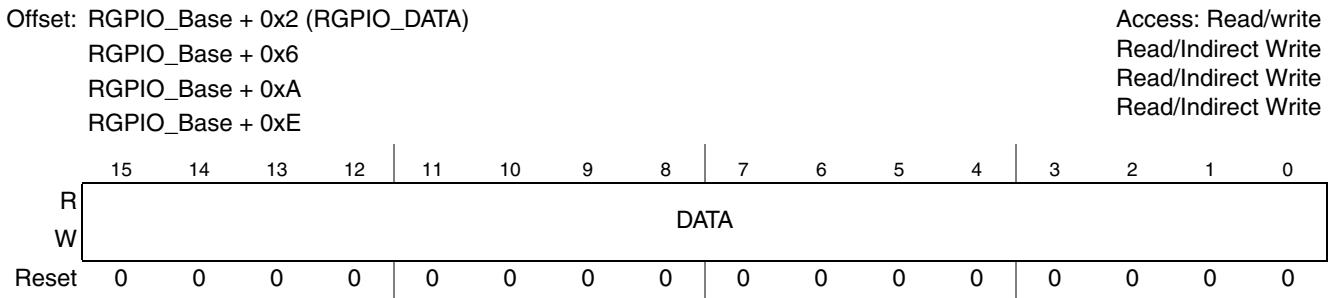


Figure 10-3. RGPIO Data Register (RGPIO\_DATA)

Table 10-6. RGPIO\_DATA Field Descriptions

Field	Description
15–0 DATA	RGPIO data. 0 A properly-enabled RGPIO output pin is driven with a logic 0, or a properly-enabled RGPIO input pin was read as a logic 0 1 A properly-enabled RGPIO output pin is driven with a logic 1, or a properly-enabled RGPIO input pin was read as a logic 1

### 10.3.3 RGPIO Pin Enable (RGPIO\_ENB)

The RGPIO\_ENB register configures the corresponding package pin as a RGPIO pin instead of the normal GPIO pin mapped onto the peripheral bus.

The RGPIO\_ENB register is read/write. At reset, all bits in the RGPIO\_ENB are cleared, disabling the RGPIO functionality.

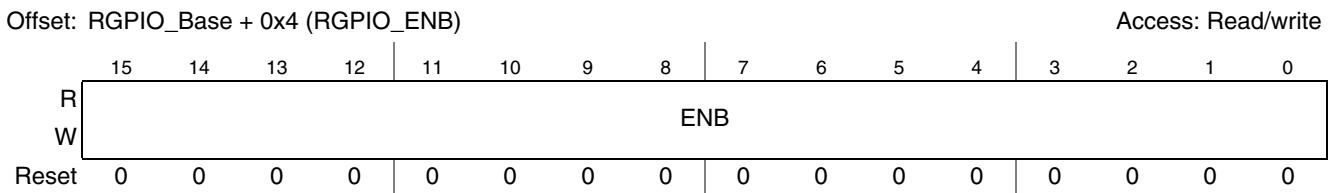


Figure 10-4. RGPIO Enable Register (RGPIO\_ENB)

Table 10-7. RGPIO\_ENB Field Descriptions

Field	Description
15–0 ENB	Enable pin for RGPIO 0 The corresponding package pin is configured for use as a normal GPIO pin, not a RGPIO 1 The corresponding package pin is configured for use as a RGPIO pin

### 10.3.4 RGPIO Clear Data (RGPIO\_CLR)

The RGPIO\_CLR register provides a mechanism to clear specific bits in the RGPIO\_DATA by performing a simple write. Clearing a bit in RGPIO\_CLR clears the corresponding bit in the RGPIO\_DATA register.

Setting it has no effect. The GPIO\_CLR register is write-only; reads of this address return the GPIO\_DATA register.

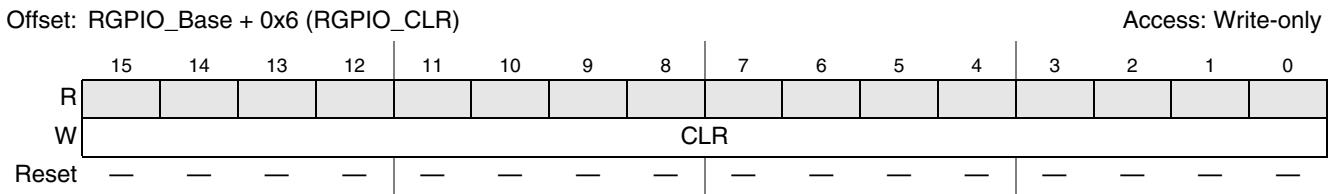


Figure 10-5. GPIO Clear Data Register (GPIO\_CLR)

Table 10-8. GPIO\_CLR Field Descriptions

Field	Description
15–0 CLR	Clear data bit 0 Clears the corresponding bit in the GPIO_DATA register 1 No effect

### 10.3.5 GPIO Set Data (GPIO\_SET)

The GPIO\_SET register provides a mechanism to set specific bits in the GPIO\_DATA register by performing a simple write. Setting a bit in GPIO\_SET asserts the corresponding bit in the GPIO\_DATA register. Clearing it has no effect. The GPIO\_SET register is write-only; reads of this address return the GPIO\_DATA register.

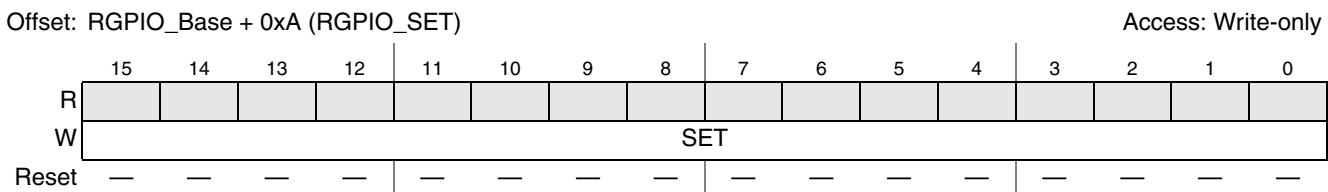


Figure 10-6. GPIO Set Data Register (GPIO\_SET)

Table 10-9. GPIO\_SET Field Descriptions

Field	Description
15–0 SET	Set data bit 0 No effect 1 Sets the corresponding bit in the GPIO_DATA register

### 10.3.6 GPIO Toggle Data (GPIO\_TOG)

The GPIO\_TOG register provides a mechanism to invert (toggle) specific bits in the GPIO\_DATA register by performing a simple write. Setting a bit in GPIO\_TOG inverts the corresponding bit in the GPIO\_DATA register. Clearing it has no effect. The GPIO\_TOG register is write-only; reads of this address return the GPIO\_DATA register.

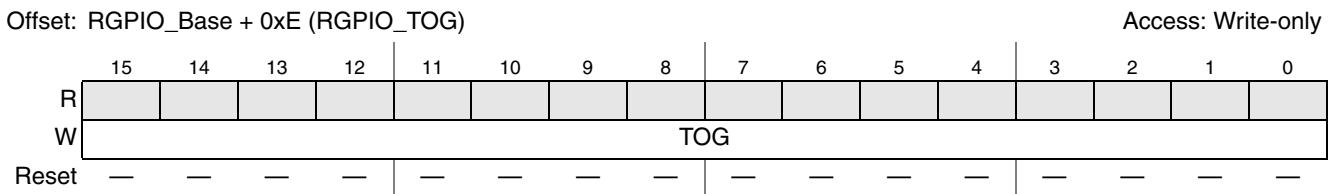


Figure 10-7. RGPIO Toggle Data Register (RGPIO\_TOG)

Table 10-10. RGPIO\_TOG Field Descriptions

Field	Description
15–0 TOG	Toggle data. 0 No effect 1 Inverts the corresponding bit in RGPIO_DATA

## 10.4 Functional Description

The RGPI0 module is a relatively-simple design with its behavior controlled by the program-visible registers defined within its programming model.

The RGPI0 module is connected to the processor's local two-stage pipelined bus with the stages of the ColdFire core's operand execution pipeline (OEP) mapped directly onto the bus. This structure allows the processor access to the RGPI0 module for single-cycle pipelined reads and writes with a zero wait-state response (as viewed in the system bus data phase stage).

## 10.5 Initialization Information

The reset state of the RGPI0 module disables the entire 16-bit data port. Prior to using the RGPI0 port, software typically:

- Enables the appropriate pins in RGPI0\_ENB
- Configures the pin direction in RGPI0\_DIR
- Defines the contents of the data register (RGPI0\_DATA)

## 10.6 Application Information

This section examines the relative performance of the RGPI0 output pins for two simple applications

- The processor executes a loop to toggle an output pin for a specific number of cycles, producing a square-wave output
- The processor transmits a 16-bit message using a three-pin SPI-like interface with a serial clock, serial chip select, and serial data bit.

In both applications, the relative speed of the GPIO output is presented as a function of the location of the output bit (RGPI0 versus peripheral bus GPIO).

## 10.6.1 Application 1: Simple Square-Wave Generation

In this example, several different instruction loops are executed, each generating a square-wave output with a 50% duty cycle. For this analysis, the executed code is mapped into the processor's RAM. This configuration is selected to remove any jitter from the output square wave caused by the limitations defined by the two-cycle flash memory accesses and restrictions on the initiation of a flash access. The following instruction loops were studied:

- **BCHG\_LOOP** — In this loop, a bit change instruction was executed using the GPIO data byte as the operand. This instruction performs a read-modify-write operation and inverts the addressed bit. A pulse counter is decremented until the appropriate number of square-wave pulses have been generated.
- **SET+CLR\_LOOP** — For this construct, two store instructions are executed: one to set the GPIO data pin and another to clear it. Single-cycle NOP instructions (the tpf opcode) are included to maintain the 50% duty cycle of the generated square wave. The pulse counter is decremented until the appropriate number of square-wave pulse have been generated.

The square-wave output frequency was measured and the relative performance results are presented in [Table 10-11](#). The relative performance is stated as a fraction of the processor's operating frequency, defined as  $f$  MHz. The performance of the BCHG loop operating on a GPIO output is selected as the reference.

**Table 10-11. Square-Wave Output Performance**

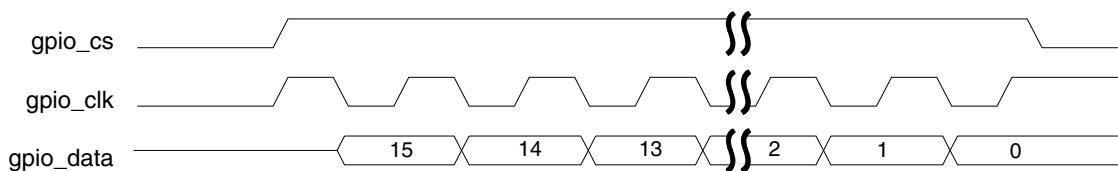
Loop	Peripheral Bus-mapped GPIO			RGPIO		
	Sq-Wave Frequency	Frequency @ CPU $f = 50$ MHz	Relative Speed	Sq-Wave Frequency	Frequency @ CPU $f = 50$ MHz	Relative Speed
bchg	$(1/24) \times f$ MHz	2.083 MHz	1.00x	$(1/14) \times f$ MHz	3.571 MHz	1.71x
set+clr (+toggle)	$(1/12) \times f$ MHz	4.167 MHz	2.00x	$(1/8) \times f$ MHz	6.250 MHz	3.00x

### NOTE

The square-wave frequency is measured from rising-edge to rising-edge, where the output wave has a 50% duty cycle.

## 10.6.2 Application 2: 16-bit Message Transmission using SPI Protocol

In this second example, a 16-bit message is transmitted using three programmable output pins. The output pins include a serial clock, an active-high chip select, and the serial data bit. The software is configured to sample the serial data bit at the rising-edge of the clock with the data sent in a most-significant to least-significant bit order. The resulting 3-bit output is shown in [Figure 10-8](#).



**Figure 10-8. GPIO SPI Example Timing Diagram**

For this example, the processing of the SPI message is considerably more complex than the generation of a simple square wave of the previous example. The code snippet used to extract the data bit from the message and build the required GPIO data register writes is shown in [Figure 10-9](#).

```
# subtest: send a 16-bit message via a SPI interface using a RGPIO

# the SPI protocol uses a 3-bit value: clock, chip-select, data
# the data is centered around the rising-edge of the clock

        align 16
send_16b_spi_message_rgpio:
00510: 4fef fff4          lea    -12(%sp),%sp      # allocate stack space
00514: 48d7 008c          movml. &0x8c,(%sp)     # save d2,d3,d7
00518: 3439 0080 0582      mov.w  RAM_BASE+message2,%d2  # get 16-bit message
0051e: 760f                movq.l &15,%d3       # static shift count
00520: 7e10                movq.l &16,%d7       # message bit length
00522: 207c 00c0 0003      mov.l  &RGPIO_DATA+1,%a0  # pointer to low-order data byte
00528: 203c 0000 ffff      mov.l  &0xffff,%d0      # data value for _ENB and _DIR regs
0052e: 3140 fffd          mov.w  %d0,-3(%a0)    # set RGPIO_DIR register
00532: 3140 0001          mov.w  %d0,1(%a0)    # set RGPIO_ENB register

00536: 223c 0001 0000      mov.l  &0x10000,%d1    # d1[17:16] = {clk, cs}
0053c: 2001                mov.l  %d1,%d0       # copy into temp reg
0053e: e6a8                lsr.l  %d3,%d0       # align in d0[2:0]
00540: 5880                addq.l &4,%d0       # set clk = 1
00542: 1080                mov.b  %d0,(%a0)    # initialize data
00544: 6002                bra.b  L%1
                                align 4

L%1:
00548: 3202                mov.w  %d2,%d1      # d1[17:15] = {clk, cs, data}
0054a: 2001                mov.l  %d1,%d0      # copy into temp reg
0054c: e6a8                lsr.l  %d3,%d0      # align in d0[2:0]
0054e: 1080                mov.b  %d0,(%a0)    # transmit data with clk = 0
00550: 5880                addq.l &4,%d0      # force clk = 1
00552: e38a                lsl.l  &1,%d2      # d2[15] = new message data bit
00554: 51fc                tpf
00556: 51fc                tpf
00558: 51fc                tpf
0055a: 51fc                tpf
0055c: 1080                mov.b  %d0,(%a0)    # transmit data with clk = 1
0055e: 5387                subq.l &1,%d7      # decrement loop counter
00560: 66e6                bne.b L%1

00562: c0bc 0000 ffff      and.l  &0xffff5,%d0    # negate chip-select
00568: 1080                mov.b  %d0,(%a0)    # update gpio

0056a: 4cd7 008c          movml. (%sp),&0x8c     # restore d2,d3,d7
0056e: 4fef 000c          lea    12(%sp),%sp      # deallocate stack space
00572: 4e75                rts
```

**Figure 10-9. GPIO SPI Code Example**

The resulting SPI performance, as measured in the effective Mbps transmission rate for the 16-bit message, is shown in [Table 10-12](#).

**Table 10-12. Emulated SPI Performance using GPIO Outputs**

Peripheral Bus-mapped GPIO		RGPIO	
SPI Speed @ CPU $f = 50$ MHz	Relative Speed	SPI Speed @ CPU $f = 50$ MHz	Relative Speed
2.063 Mbps	1.00x	3.809 Mbps	1.29x

# Chapter 11

## Mini-FlexBus

### 11.1 Introduction

This chapter describes external bus data transfer operations and error conditions. It describes transfers initiated by the ColdFire processor (or any other bus master) and includes detailed timing diagrams showing the interaction of signals in supported bus operations.

The Mini-FlexBus is a subset of the FlexBus module found on other ColdFire microprocessors. The Mini-FlexBus minimizes package pin-outs while maintaining a high level of configurability and functionality.

#### NOTE

- In this chapter, unless otherwise noted, clock refers to the FB\_CLK used for the external bus ( $f_{sys}$ ).
- Use pin mux control registers from [Section 2.3, “Pin Mux Controls”](#) to assign Mini-FlexBus signals to the MCF51CN128 package pins.
- Most pin functions default to GPIO and must be software configured before using Mini-FlexBus.

#### 11.1.1 Overview

A multi-function external bus interface called the Mini-FlexBus interface controller is provided on the device with basic functionality of interfacing to slave-only devices. It can be directly connected to the following asynchronous or synchronous devices with little or no additional circuitry:

- External ROMs
- Flash memories
- Programmable logic devices
- Other simple target (slave) devices

For asynchronous devices, a simple chip-select based interface can be used.

The Mini-FlexBus interface has up to two general purpose chip-selects,  $\overline{FB\_CS}[1:0]$ . The actual number of chip selects available depends upon the device and its pin configuration.

## 11.1.2 Features

Key Mini-FlexBus features include:

- Two independent, user-programmable chip-select signals ( $\overline{\text{FB\_CS}}[1:0]$ ) that can interface with external SRAM, PROM, EPROM, EEPROM, flash, and other peripherals
- 8- and 16-bit port sizes with configuration for multiplexed or non-multiplexed address and data buses
- Byte-, word-, longword-, and 16-byte line-sized transfers
- Programmable address-setup time with respect to the assertion of chip select
- Programmable address-hold time with respect to the negation of chip select and transfer direction

## 11.1.3 Modes of Operation

The external interface is a configurable multiplexed bus set to one of the following modes:

- Up to a 20-bit address (non-multiplexed) with 8-bit data
- Up to a 20-bit address (multiplexed) with 16-bit data (write masking of upper/lower bytes not supported)
- Up to a 20-bit address (multiplexed) with 8-bit data

## 11.2 External Signals

This section describes the external signals involved in data-transfer operations.

**Table 11-1. Mini-FlexBus Signal Summary**

Signal Name	I/O	Description
$\text{FB\_A}[19:0]$	I/O	In a non-multiplexed configuration, this is the address bus. In a multiplexed configuration this bus is the address/data bus, $\text{FB\_AD}[19:0]$ .
$\text{FB\_D}[7:0]$	I/O	In a non-multiplexed configuration, this is the data bus. In multiplexed configurations, this bus is not used.
$\overline{\text{FB\_CS}}[1:0]$	O	General purpose chip-selects. In multiplexed mode, only $\overline{\text{FB\_CS}0}$ is available. $\overline{\text{FB\_CS}1}$ is multiplexed with $\text{FB\_ALE}$ on a configurable package pin.
$\overline{\text{FB\_OE}}$	O	Output enable
$\text{FB\_R/W}$	O	Read/write. 1 = Read, 0 = Write
$\text{FB\_ALE}$	O	Address latch enable. This signal is multiplexed with $\overline{\text{FB\_CS}1}$ on a configurable package pin.

### 11.2.1 Address and Data Buses ( $\text{FB\_A}[19:0]$ , $\text{FB\_D}[7:0]$ , $\text{FB\_AD}[19:0]$ )

In non-multiplexed mode, the  $\text{FB\_A}[19:0]$  and  $\text{FB\_D}[7:0]$  buses carry the address and data, respectively.

In multiplexed mode, the FB\_AD[19:0] bus carries the address and data. The full 20-bit address is driven on the first clock of a bus cycle (address phase). Following the first clock, the data is driven on the bus (data phase). During the data phase, the address continues driving on the pins not used for data. For example, in 16-bit mode the address continues driving on FB\_AD[19:16] and in 8-bit mode the address continues driving on FB\_AD[19:8].

### 11.2.2 Chip Selects (FB\_CS[1:0])

The chip-select signal indicates which device is selected. A particular chip-select asserts when the transfer address is within the device's address space, as defined in the base- and mask-address registers. The actual number of chip selects available depends upon the pin configuration.

### 11.2.3 Output Enable (FB\_OE)

The output enable signal ( $\overline{FB\_OE}$ ) is sent to the interfacing memory and/or peripheral to enable a read transfer.  $\overline{FB\_OE}$  is only asserted during read accesses when a chip select matches the current address decode.

### 11.2.4 Read/Write (FB\_R/W)

The processor drives the FB\_R/W signal to indicate the current bus operation direction. It is driven high during read bus cycles and low during write bus cycles.

### 11.2.5 Address Latch Enable (FB\_ALE)

The assertion of FB\_ALE indicates that the device has begun a bus transaction and the address and attributes are valid. FB\_ALE is asserted for one bus clock cycle. FB\_ALE may be used externally to capture the bus transfer address ([Figure 11-7](#)).

## 11.3 Memory Map/Register Definition

The following tables describe the registers and bit meanings for configuring chip-select operation. [Table 11-2](#) shows the chip-select register memory map.

The actual number of chip select registers available depends upon the device and its pin configuration. If the device does not support certain chip select signals or the pin is not configured for a chip-select function, then that corresponding set of chip-select registers has no effect on an external pin.

#### NOTE

You must set CSMR0[V] before the chip select registers take effect.

**Table 11-2. Mini-FlexBus Chip Select Memory Map**

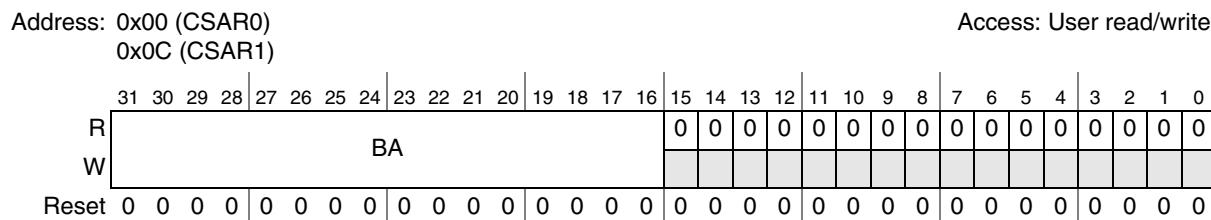
Offset	Register	Width (bits)	Access	Reset Value	Section/ Page
0x00 0x0C	Chip-Select Address Register (CSAR $n$ ) $n = 0 - 1$	32	R/W	0x0000_0000	<a href="#">11.3.1/11-4</a>
0x04 0x10	Chip-Select Mask Register (CSMR $n$ ) $n = 0 - 1$	32	R/W	0x0000_0000	<a href="#">11.3.2/11-4</a>
0x08 0x14	Chip-Select Control Register (CSCR $n$ ) $n = 0 - 1$	32	R/W	See Section	<a href="#">11.3.3/11-5</a>

### 11.3.1 Chip-Select Address Registers (CSAR0 – CSAR1)

The CSAR $n$  registers specify the chip-select base addresses.

#### NOTE

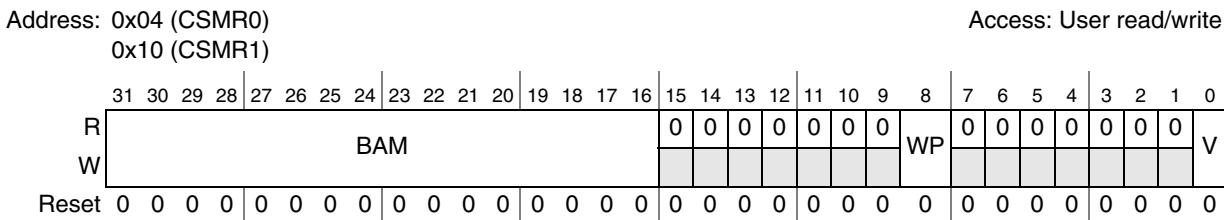
The only applicable address range for which the chip-selects can be active are 0x(00)40\_0000 – 0x(00)7F\_FFFF. Set the CSAR $n$  and CSMR $n$  registers appropriately before accessing this region.

**Figure 11-1. Chip-Select Address Registers (CSAR $n$ )****Table 11-3. CSAR $n$  Field Descriptions**

Field	Description
31–16 BA	Base address. Defines the base address for memory dedicated to chip-select FB_CSn. BA is compared to bits 31–16 on the internal address bus to determine if chip-select memory is being accessed.
15–0	Reserved, must be cleared.

### 11.3.2 Chip-Select Mask Registers (CSMR0 – CSMR1)

CSMR $n$  registers specify the address mask and allowable access types for the respective chip-selects.

**Figure 11-2. Chip-Select Mask Registers (CSMR $n$ )**

**Table 11-4. CSMR $n$  Field Descriptions**

Field	Description
31–16 BAM	Base address mask. Defines the chip-select block size by masking address bits. Setting a BAM bit causes the corresponding CSAR bit to be a don't care 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 $\overline{FB\_CS}_n$ is $2^n$ ; $n = (\text{number of bits set in respective } \text{CSMR}[\text{BAM}]) + 16$ . For example, if CSAR0 equals 0x0040 and CSMR0[BAM] equals 0x000, $\overline{FB\_CS}_0$ addresses two discontinuous 64 KB memory blocks: one from 0x40_0000 – 0x40_FFFF and one from 0x48_0000 – 0x48_FFFF. Likewise, for $\overline{FB\_CS}_0$ to access 2 MB of address space starting at location 0x40_0000, $\overline{FB\_CS}_1$ must begin at the next byte after $\overline{FB\_CS}_0$ for a 1 MB address space. Therefore, CSAR0 equals 0x0040, CSMR0[BAM] equals 0x001F, CSAR1 equals 0x0060, and CSMR1[BAM] equals 0x000F.
15–9	Reserved, must be cleared.
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 CSAR $n$ [WP] is set results in a bus error termination of the internal cycle and no external cycle. 0 Read and write accesses are allowed 1 Only read accesses are allowed
7–1	Reserved, must 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 bit is set. Reset clears each CSMR $n$ [V]. <b>Note:</b> At reset, no chip-select can be used until the CSMR0[V] is set. Afterward, $\overline{FB\_CS}[1:0]$ functions as programmed. 0 Chip-select invalid 1 Chip-select valid

### 11.3.3 Chip-Select Control Registers (CSCR0 – CSCR1)

Each CSCR $n$  controls the auto-acknowledge, address setup and hold times, port size, and number of wait states.

Address: 0x08 (CSCR0) 0x14 (CSCR1)												Access: User read/write					
R	31	30	29	28	27	26	25	24	23	22	21	20	ASET		RDAH		WRAH
W																	
Reset: CSCR0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	
Reset: CSCR1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
R	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
W	WS					MUX	AA	PS		0	0		0	0	0	0	
Reset: CSCR0	1	1	1	1	1	1	0	1	0	0	0	0	0	0	0	0	
Reset: CSCR1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

**Figure 11-3. Chip-Select Control Registers (CSCR $n$ )**

**Table 11-5. CSCR<sub>n</sub> Field Descriptions**

Field	Description										
31–22	Reserved, must be cleared										
21–20 ASET	<p>Address setup. This field controls the assertion of the chip-select with respect to assertion of a valid address and attributes. The address and attributes are considered valid at the same time FB_ALE asserts.</p> <p>00 Assert <math>\overline{\text{FB\_CS}_n}</math> on first rising clock edge after address is asserted. (Default <math>\overline{\text{FB\_CS}1}</math>)</p> <p>01 Assert <math>\overline{\text{FB\_CS}_n}</math> on second rising clock edge after address is asserted.</p> <p>10 Assert <math>\overline{\text{FB\_CS}_n}</math> on third rising clock edge after address is asserted.</p> <p>11 Assert <math>\overline{\text{FB\_CS}_n}</math> on fourth rising clock edge after address is asserted. (Default <math>\overline{\text{FB\_CS}0}</math>)</p>										
19–18 RDAH	<p>Read address hold or deselect. This field controls the address and attribute hold time after the termination during a read cycle that hits in the chip-select address space.</p> <p><b>Note:</b> The hold time applies only at the end of a transfer. Therefore, during a transfer to a port size smaller than the transfer size, the hold time is only added after the last bus cycle.</p> <p>The number of cycles the address and attributes are held after <math>\overline{\text{FB\_CS}_n}</math> negation depends on the value of CSCR<sub>n</sub>[AA] as shown below.</p> <table border="1"> <thead> <tr> <th>RDAH</th> <th>AA = 1</th> </tr> </thead> <tbody> <tr> <td>00 (<math>\overline{\text{FB\_CS}1}</math> Default)</td> <td>0 cycles</td> </tr> <tr> <td>01</td> <td>1 cycles</td> </tr> <tr> <td>10</td> <td>2 cycles</td> </tr> <tr> <td>11 (<math>\overline{\text{FB\_CS}0}</math> Default)</td> <td>3 cycles</td> </tr> </tbody> </table>	RDAH	AA = 1	00 ( $\overline{\text{FB\_CS}1}$ Default)	0 cycles	01	1 cycles	10	2 cycles	11 ( $\overline{\text{FB\_CS}0}$ Default)	3 cycles
RDAH	AA = 1										
00 ( $\overline{\text{FB\_CS}1}$ Default)	0 cycles										
01	1 cycles										
10	2 cycles										
11 ( $\overline{\text{FB\_CS}0}$ Default)	3 cycles										
17–16 WRAH	<p>Write address hold or deselect. This field controls the address, data, and attribute hold time after the termination of a write cycle that hits in the chip-select address space.</p> <p><b>Note:</b> The hold time applies only at the end of a transfer. Therefore, during a transfer to a port size smaller than the transfer size, the hold time is only added after the last bus cycle.</p> <p>00 Hold address and attributes one cycle after <math>\overline{\text{FB\_CS}_n}</math> negates on writes. (Default <math>\overline{\text{FB\_CS}1}</math>)</p> <p>01 Hold address and attributes two cycles after <math>\overline{\text{FB\_CS}_n}</math> negates on writes.</p> <p>10 Hold address and attributes three cycles after <math>\overline{\text{FB\_CS}_n}</math> negates on writes.</p> <p>11 Hold address and attributes four cycles after <math>\overline{\text{FB\_CS}_n}</math> negates on writes. (Default <math>\overline{\text{FB\_CS}0}</math>)</p>										
15–10 WS	Wait states. The number of wait states inserted after $\overline{\text{FB\_CS}_n}$ asserts and before an internal transfer acknowledge is generated (WS = 0 inserts zero wait states, WS = 0x3F inserts 63 wait states).										
9 MUX	Multiplexed mode. Selects between multiplexed and non-multiplexed address/data bus. 0 Non-multiplexed configuration. Address information is driven on FB_ADn and data is read/written on FB_dn. 1 Non-multiplexed configuration. Address information is driven on FB_ADn, and low-order address lines (FB_AD[7:0] for byte port size or FB_AD[15:0] for word port size) must be latched using the falling edge of FB_ALE as the latch enable. Data is read/written on FB_AD[7:0] for byte port size and FB_AD[15:0] for word port size.										
8 AA	<p>Auto-acknowledge enable. Determines the assertion of the internal transfer acknowledge for accesses specified by the chip-select address. This bit must be set.</p> <p>0 Reserved</p> <p>1 Internal transfer acknowledge is asserted as specified by WS</p> <p><b>Note:</b> This bit must be set, since only internal termination is supported by the Mini-FlexBus.</p>										

**Table 11-5. CSCR $n$  Field Descriptions (continued)**

Field	Description
7–6 PS	Port size. Specifies the data port width associated with each chip-select. It determines where data is driven during write cycles and where data is sampled during read cycles. 00 Reserved 01 8-bit port size. Valid data sampled and driven on FB_D[7:0] 1x 16-bit port size. Valid data sampled and driven on FB_AD[15:0]. Only supported in multiplexed mode.
5–0	Reserved, must be cleared.

## 11.4 Functional Description

### 11.4.1 Chip-Select Operation

Each chip-select has a dedicated set of registers for configuration and control:

- Chip-select address registers (CSAR $n$ ) control the base address space of the chip-select. See [Section 11.3.1, “Chip-Select Address Registers \(CSAR0 – CSAR1\).”](#)
- Chip-select mask registers (CSMR $n$ ) provide 16-bit address masking and access control. See [Section 11.3.2, “Chip-Select Mask Registers \(CSMR0 – CSMR1\).”](#)
- Chip-select control registers (CSCR $n$ ) provide port size, wait-state generation, address setup and hold times, and automatic acknowledge generation features. See [Section 11.3.3, “Chip-Select Control Registers \(CSCR0 – CSCR1\).”](#)

#### 11.4.1.1 General Chip-Select Operation

When a bus cycle is routed to the Mini-FlexBus, the device first compares its address with the base address and mask configurations programmed for chip-selects 0 and 1 (configured in CSCR0 – CSCR1). The results depend on if the address matches or not as shown in [Table 11-6](#).

**Table 11-6. Results of Address Comparison**

Address Matches CSAR $n$ ?	Result
Yes, one CSAR	The appropriate chip-select is asserted, generating an external bus cycle as defined in the chip-select control register. If CSMR[WP] is set and a write access is performed, the internal bus cycle terminates with a bus error, no chip select is asserted, and no external bus cycle is performed.
No	The internal bus cycle terminates with a bus error, no chip select is asserted, and no external bus cycle is performed.
Yes, multiple CSARs	The internal bus cycle terminates with a bus error, no chip select is asserted, and no external bus cycle is performed.

#### 11.4.1.2 8- and 16-Bit Port Sizing

Static bus sizing is programmable through the port size bits, CSCR[PS]. The processor always drives a 20-bit address on the FB\_AD bus regardless of the external device’s address size. The external device must

connect its address lines to the appropriate FB\_AD bits from FB\_AD0 upward. Its data bus must be connected to FB\_AD[7:0] in non-multiplexed mode (CSCR[MUX] = 0) or FB\_AD0 to FB\_AdN in multiplexed mode (CSCR[MUX] = 1) where n = 15 if CSCR[PS] = 1x or n = 7 if CSCR[PS] = 01. No bit ordering is required when connecting address and data lines to the FB\_AD bus. For example, a full 16-bit address/16-bit data device connects its addr[15:0] to FB\_AD[16:1] and data[15:0] to FB\_AD[15:0]. See [Figure 11-4](#) for a graphical connection.

### 11.4.2 Data Transfer Operation

Data transfers between the chip and other devices involve these signals:

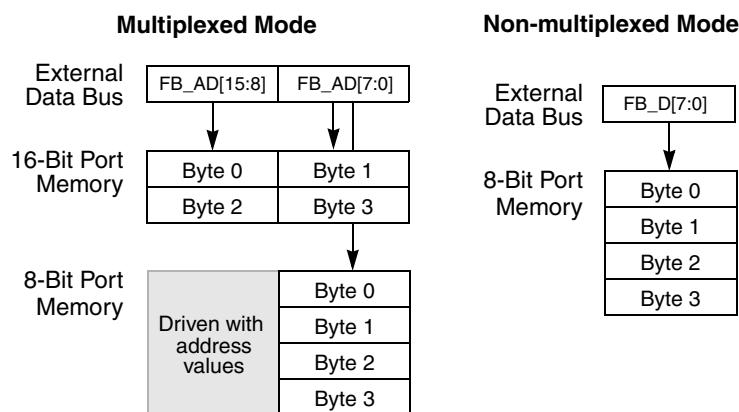
- Address/data bus (FB\_AD[19:0])
- Control signals (FB\_ALE,  $\overline{FB\_CSn}$ ,  $\overline{FB\_OE}$ )
- Attribute signals (FB\_R/W)

The address, write data, FB\_ALE,  $\overline{FB\_CSn}$ , and all attribute signals change on the rising edge of the Mini-FlexBus clock (FB\_CLK). Read data is latched into the device on the rising edge of the clock.

The Mini-FlexBus supports byte-, word-, longword-, and 16-byte (line) operand transfers and allows accesses to 8- and 16-bit data ports. Transfer parameters (address setup and hold, port size, the number of wait states for the external device being accessed, automatic internal transfer termination enable or disable) are programmed in the chip-select control registers (CSCRs). See [Section 11.3.3, “Chip-Select Control Registers \(CSCR0 – CSCR1\).”](#)

### 11.4.3 Data Byte Alignment and Physical Connections

The device aligns data transfers in Mini-FlexBus byte lanes with the number of lanes depending on the data port width. [Figure 11-4](#) shows the byte lanes that external memory connects to and the sequential transfers of a longword transfer for the supported port sizes. For example, an 8-bit memory connects to the single lane FB\_AD[7:0]. A longword transfer through this 8-bit port takes four transfers, starting with the MSB to the LSB.



**Figure 11-4. Connections for External Memory Port Sizes**

#### 11.4.4 Address/Data Bus Multiplexing

The interface supports a single 20-bit wide multiplexed address and data bus (FB\_AD[19:0]). The full 20-bit address is always driven on the first clock of a bus cycle. During the data phase, the FB\_AD[15:0] lines used for data are determined by the programmed port size for the corresponding chip select. The device continues to drive the address on any FB\_AD[15:0] lines not used for data.

The table below lists the supported combinations of address and data bus widths.

**Table 11-8. Mini-FlexBus Multiplexed Operating Modes**

Port Size & Phase		FB_AD		
		[19:16]	[15:8]	[7:0]
16-bit	Address phase	Address		
	Data phase	Address	Data	
8-bit	Address phase	Address		
	Data phase	Address	Data	

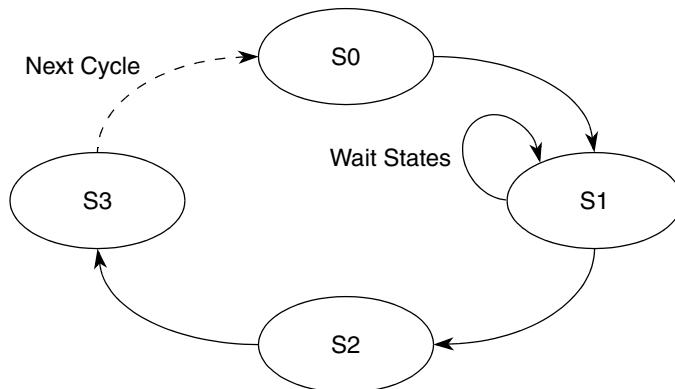
#### 11.4.5 Bus Cycle Execution

As shown in [Figure 11-7](#) and [Figure 11-9](#), basic bus operations occur in four clocks:

1. S0: At the first clock edge, the address, attributes, and FB\_ALE are driven.
2. S1:  $\overline{FB\_CSn}$  is asserted at the second rising clock edge to indicate the device selected; by that time, the address and attributes are valid and stable. FB\_ALE is negated at this edge.  
For a write transfer, data is driven on the bus at this clock edge and continues to be driven until one clock cycle after  $\overline{FB\_CSn}$  negates. For a read transfer, data is also driven into the device during this cycle.
3. S2: Read data is sampled on the third clock edge. After this edge read data can be tri-stated.
4. S3:  $\overline{FB\_CSn}$  is negated at the fourth rising clock edge. This 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.

### 11.4.5.1 Data Transfer Cycle States

An on-chip state machine controls the data-transfer operation in the device. Figure 11-5 shows the state-transition diagram for basic read and write cycles.



**Figure 11-5. Data-Transfer-State-Transition Diagram**

Table 11-9 describes the states as they appear in subsequent timing diagrams.

**Table 11-9. Bus Cycle States**

State	Cycle	Description
S0	All	The read or write cycle is initiated. On the rising clock edge, the device places a valid address on FB_AD[19:0], asserts FB_ALE, and drives FB_R/W high for a read and low for a write.
S1	All	FB_ALE is negated on the rising edge of FB_CLK, and $\overline{FB\_CS_n}$ is asserted. Data is driven on FB_AD[X:0] for writes, and FB_AD[X:0] is tristated for reads. Address continues to be driven on the FB_AD pins that are unused for data.
	Read	Data is driven by the external device before the next rising edge of FB_CLK (the rising edge that begins S2).
S2	All	$\overline{FB\_CS_n}$ is negated and the internal system bus transfer is completed.
	Read	The processor latches data on the rising clock edge entering S2. The external device can stop driving data after this edge. However, data can be driven until the end of S3 or any additional address hold cycles.
S3	All	Address, data, and FB_R/W go invalid off the rising edge of FB_CLK at the beginning of S3, terminating the read or write cycle.

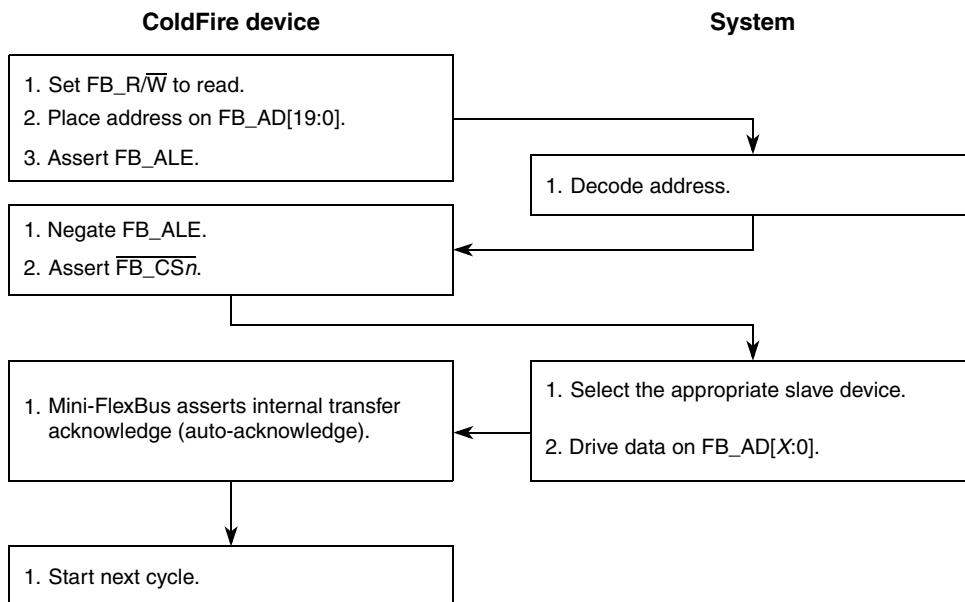
### 11.4.6 Mini-FlexBus Timing Examples

#### 11.4.6.1 Basic Read Bus Cycle

During a read cycle, the ColdFire device receives data from memory or a peripheral device. Figure 11-6 is a read cycle flowchart.

**NOTE**

Throughout this chapter FB<sub>\_</sub>AD[X:0] indicates a 16-, or 8-bit wide data bus. FB<sub>\_</sub>AD[19:X+1] is an address bus that can be 12-, or 4-bits in width.



**Figure 11-6. Read Cycle Flowchart**

The read cycle timing diagram is shown in [Figure 11-7](#).

**NOTE**

The processor drives the data lines during the first clock cycle of the transfer with the full 20-bit address. This may be ignored by standard connected devices using non-multiplexed address and data buses. However, some applications may find this feature beneficial.

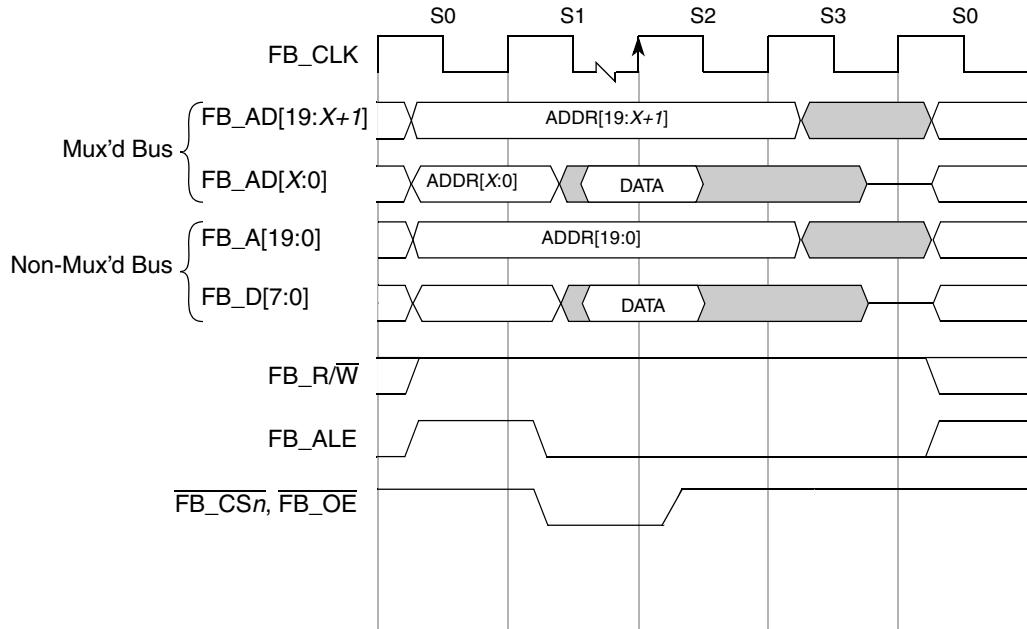


Figure 11-7. Basic Read-Bus Cycle

#### 11.4.6.2 Basic Write Bus Cycle

During a write cycle, the device sends data to memory or to a peripheral device. [Figure 11-8](#) shows the write cycle flowchart.

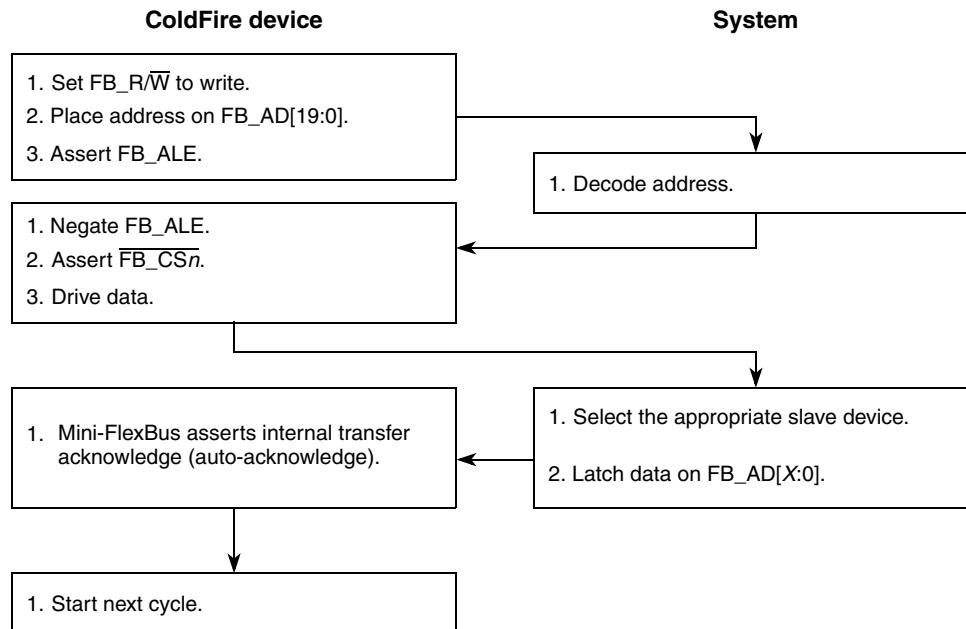


Figure 11-8. Write-Cycle Flowchart

Figure 11-9 shows the write cycle timing diagram.

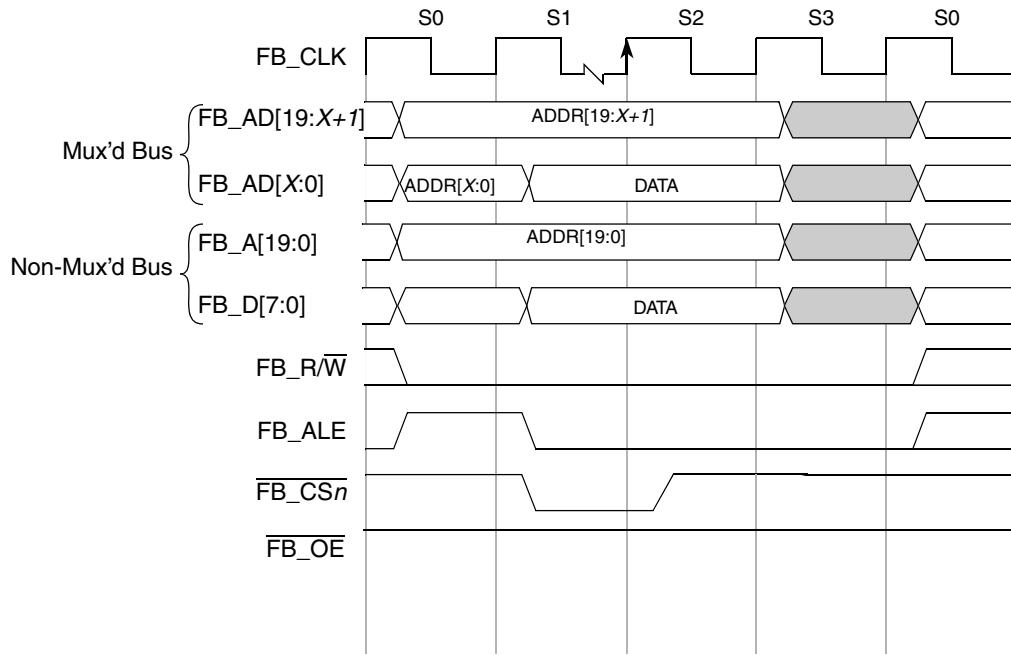
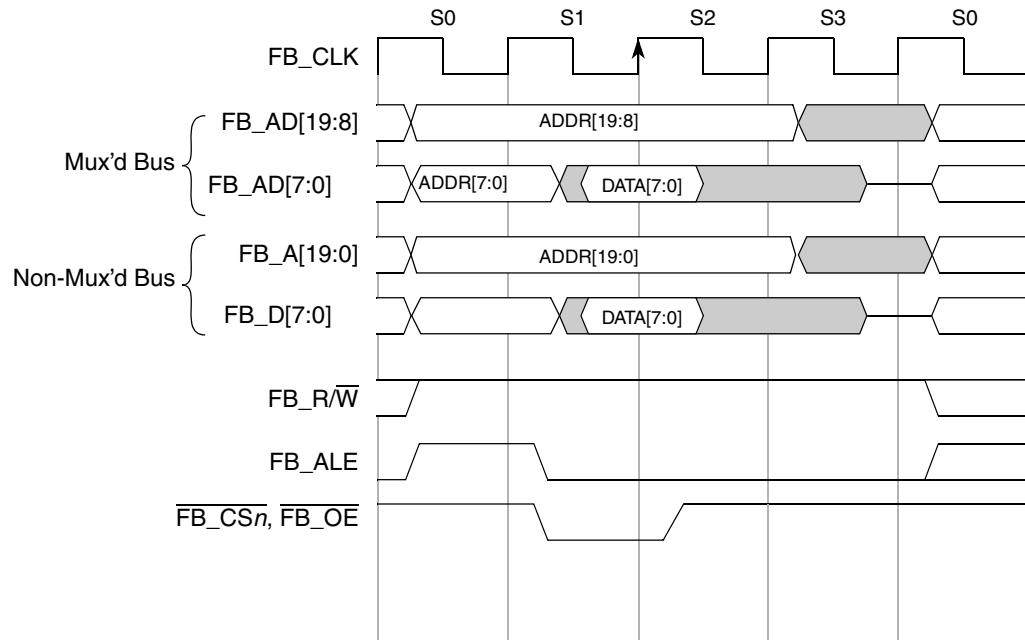


Figure 11-9. Basic Write-Bus Cycle

#### 11.4.6.3 Bus Cycle Sizing

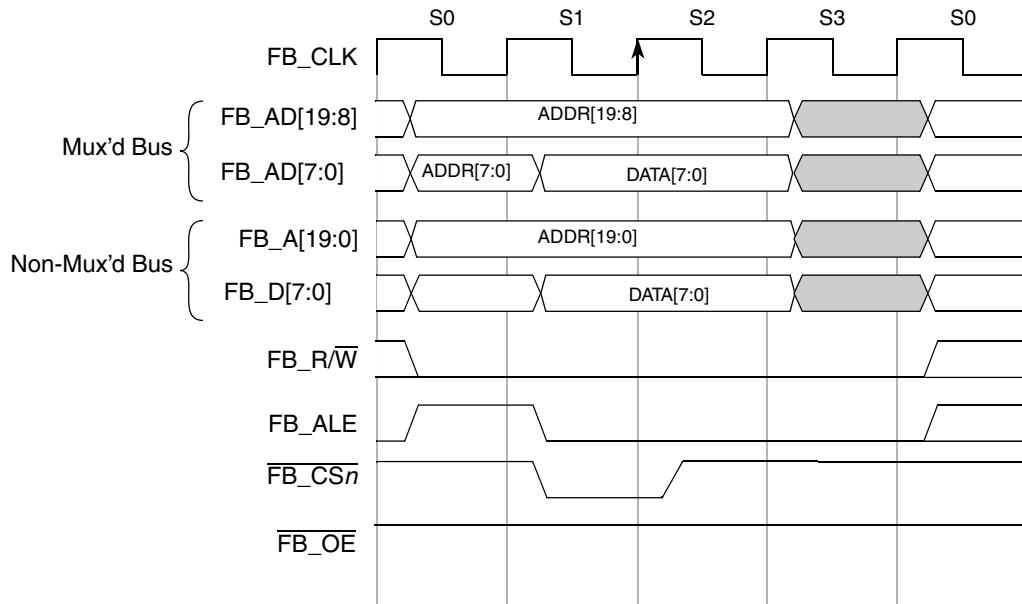
This section shows timing diagrams for various port size scenarios. Figure 11-10 illustrates the basic byte read transfer to an 8-bit device with no wait states. The address is driven on the full FB\_AD[19:8] bus in

the first clock. The device tristates FB\_AD[7:0] on the second clock and continues to drive address on FB\_AD[19:8] throughout the bus cycle. The external device returns the read data on FB\_AD[7:0].



**Figure 11-10. Single Byte-Read Transfer**

Figure 11-11 shows the similar configuration for a write transfer. The data is driven from the second clock on FB\_AD[7:0].

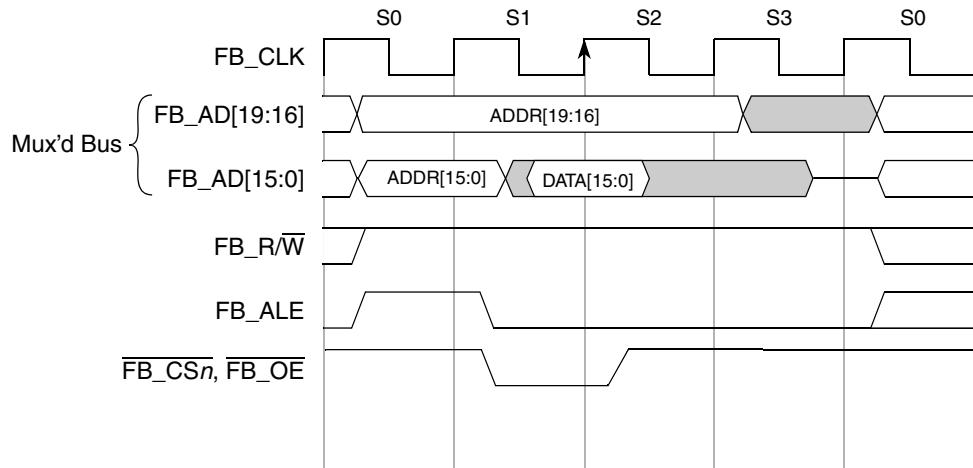


**Figure 11-11. Single Byte-Write Transfer**

[Figure 11-12](#) illustrates the basic word read transfer to a 16-bit device with no wait states. The address is driven on the full FB\_AD[19:0] bus in the first clock. The device tristates FB\_AD[15:0] on the second clock and continues to drive the address on FB\_AD[19:16] throughout the bus cycle. The external device returns the read data on FB\_AD[15:0].

### NOTE

In non-multiplexed mode, the Mini-FlexBus does not support connection to a 16-bit device.

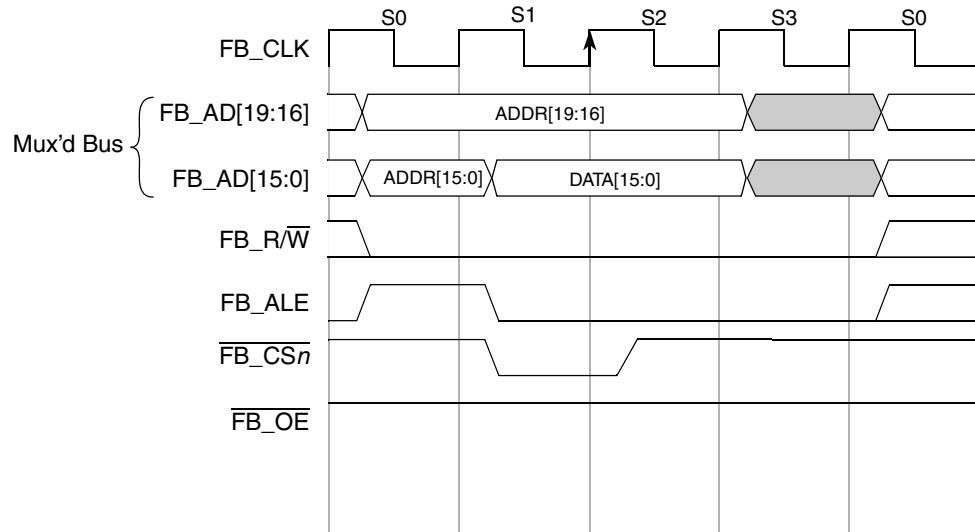


**Figure 11-12. Single Word-Read Transfer**

[Figure 11-13](#) shows the similar configuration for a write transfer. The data is driven from the second clock on FB\_ADR[15:0].

### NOTE

In non-multiplexed mode, the Mini-FlexBus does not support connection to a 16-bit device.



**Figure 11-13. Single Word-Write Transfer**

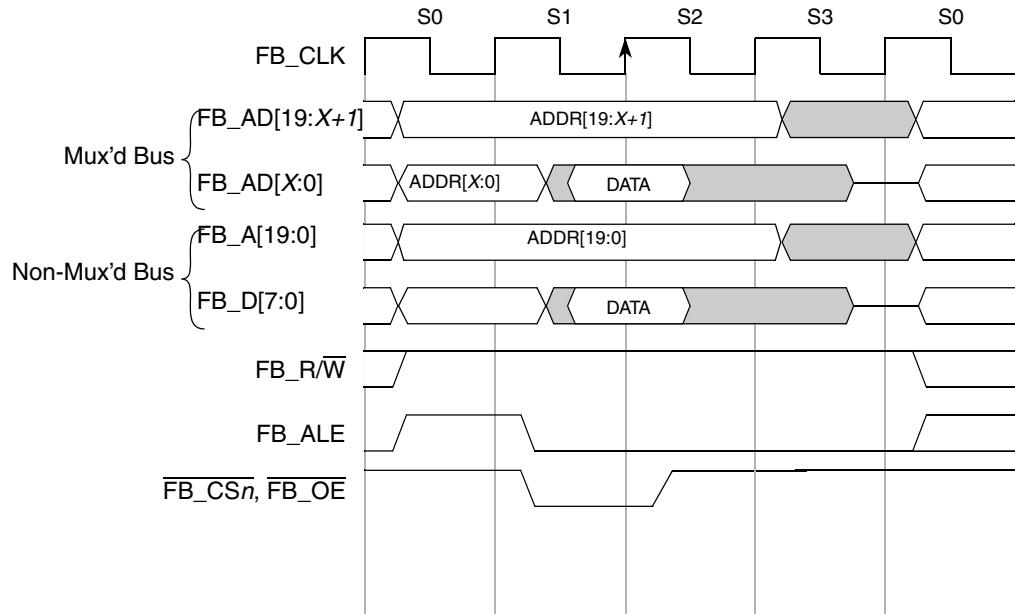
#### 11.4.6.4 Timing Variations

The Mini-FlexBus module has several features that can change the timing characteristics of a basic read-or write-bus cycle to provide additional address setup, address hold, and time for a device to provide or latch data.

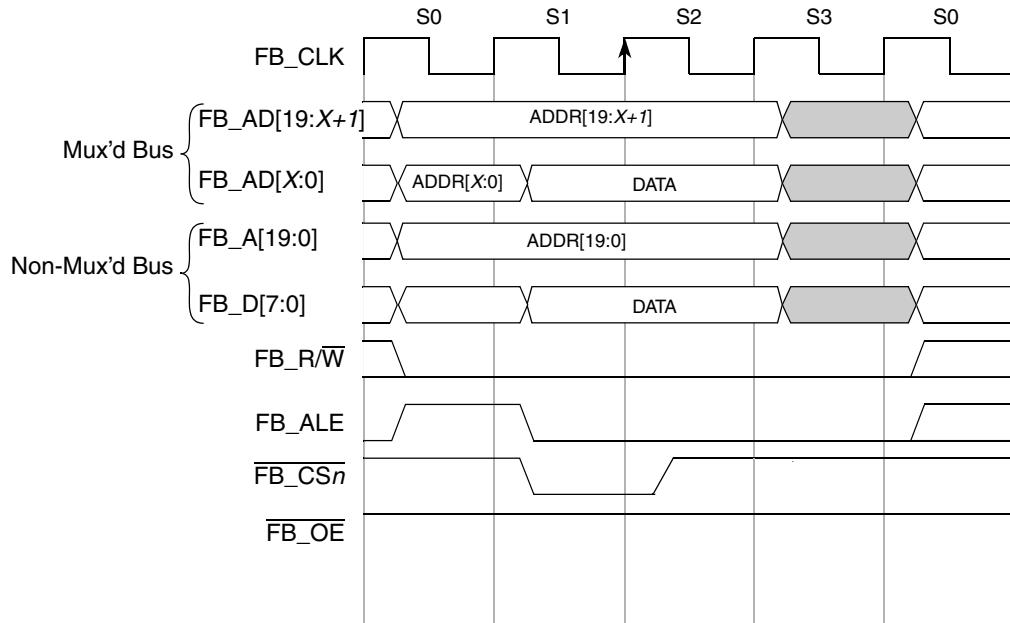
##### 11.4.6.4.1 Wait States

Wait states can be inserted before each beat of a transfer by programming the CSCR $n$  registers. Wait states can give the peripheral or memory more time to return read data or sample write data.

Figure 11-14 and Figure 11-15 show the basic read and write bus cycles (also shown in Figure 11-7 and Figure 11-12) with the default of no wait states.

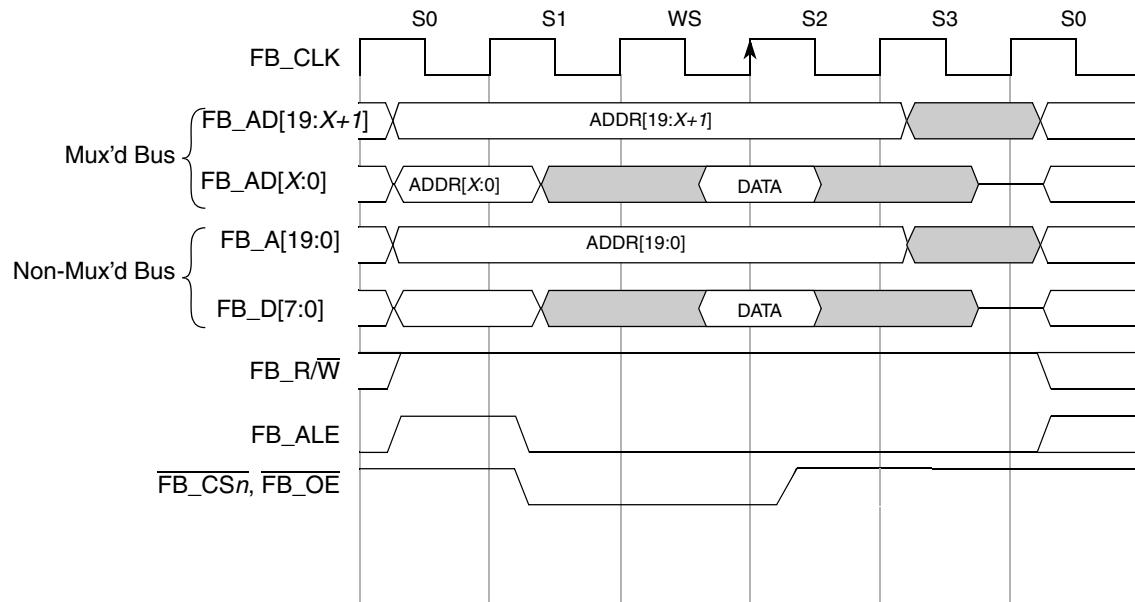


**Figure 11-14. Basic Read-Bus Cycle (No Wait States)**

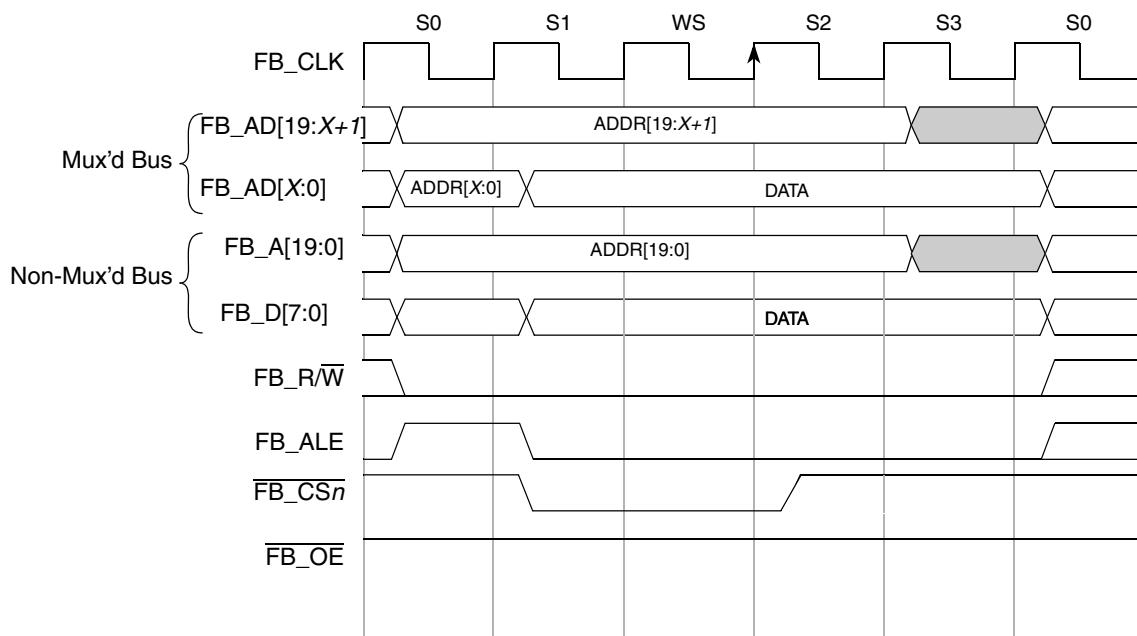


**Figure 11-15. Basic Write-Bus Cycle (No Wait States)**

If wait states are used, the S1 state repeats continuously until the the chip-select auto-acknowledge unit asserts internal transfer acknowledge. [Figure 11-16](#) and [Figure 11-17](#) show a read and write cycle with one wait state.



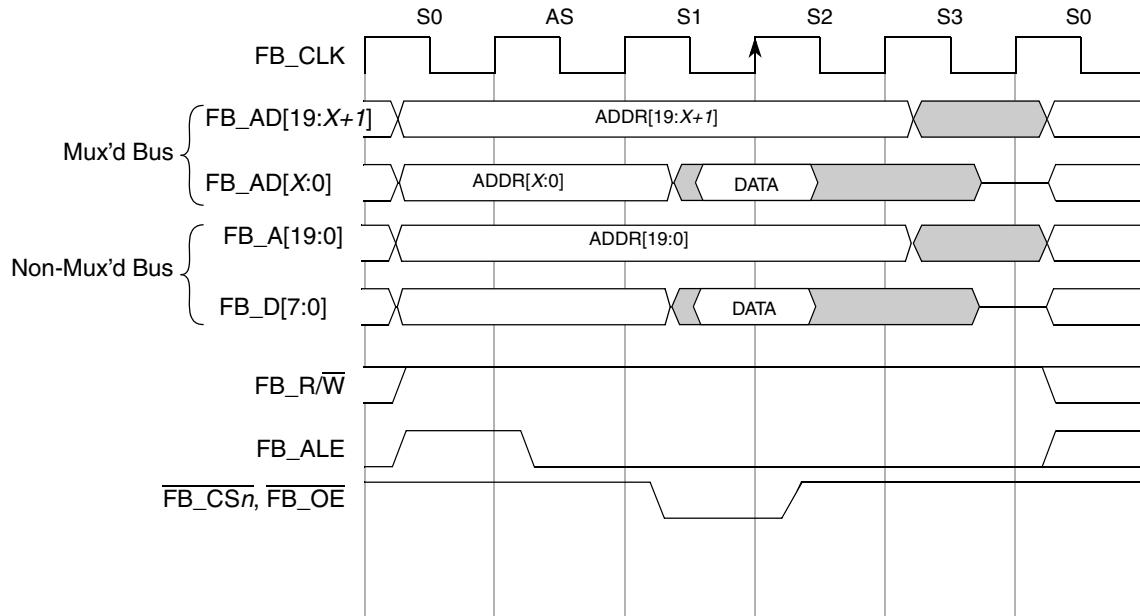
**Figure 11-16. Read-Bus Cycle (One Wait State)**



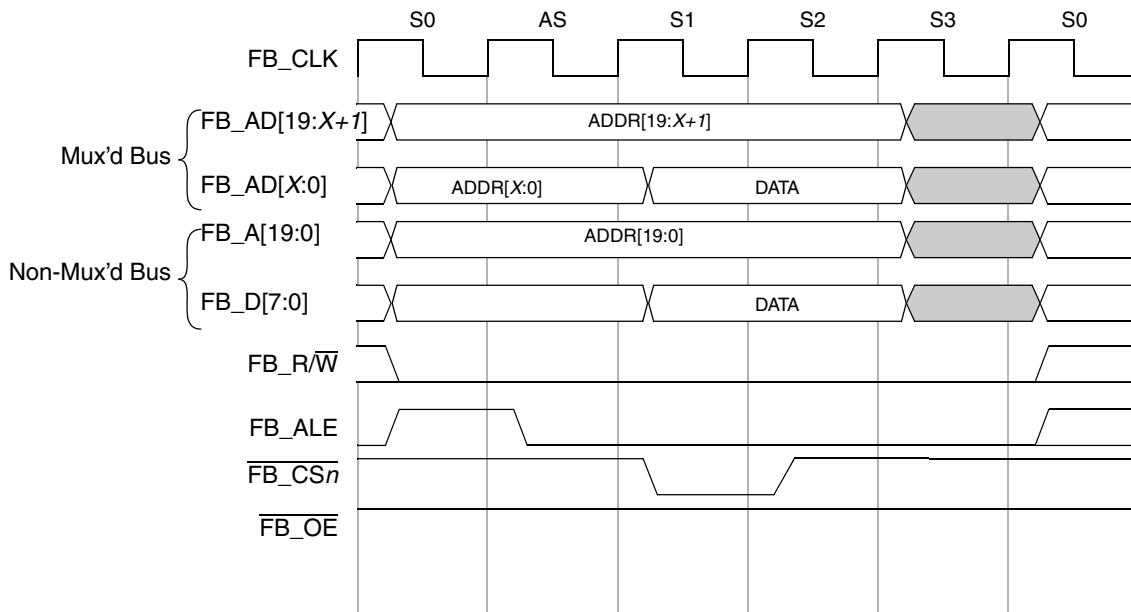
**Figure 11-17. Write-Bus Cycle (One Wait State)**

#### 11.4.6.4.2 Address Setup and Hold

The timing of the assertion and negation of the chip selects, byte selects, and output enable can be programmed on a chip-select basis. Each chip-select can be programmed to assert one to four clocks after address-latch enable (FB\_ALE) is asserted. [Figure 11-18](#) and [Figure 11-19](#) show read- and write-bus cycles with two clocks of address setup.



**Figure 11-18. Read-Bus Cycle with Two-Clock Address Setup (No Wait States)**



**Figure 11-19. Write-Bus Cycle with Two Clock Address Setup (No Wait States)**

In addition to address setup, a programmable address hold option for each chip select exists. Address and attributes can be held one to four clocks after chip-select, byte-selects, and output-enable negate. Figure 11-20 and Figure 11-21 show read and write bus cycles with two clocks of address hold.

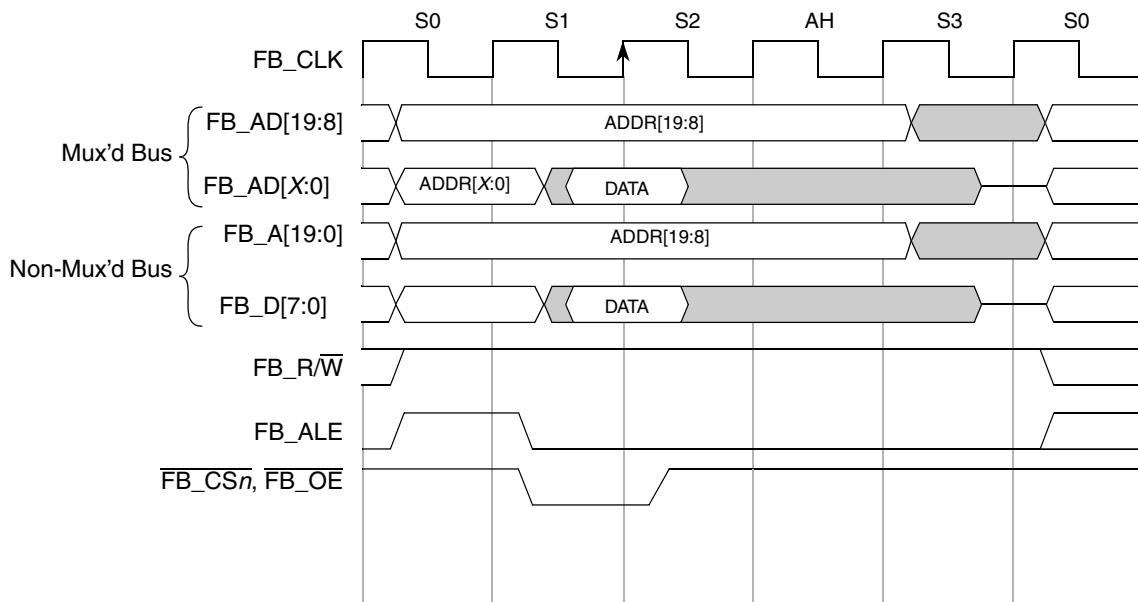


Figure 11-20. Read Cycle with Two-Clock Address Hold (No Wait States)

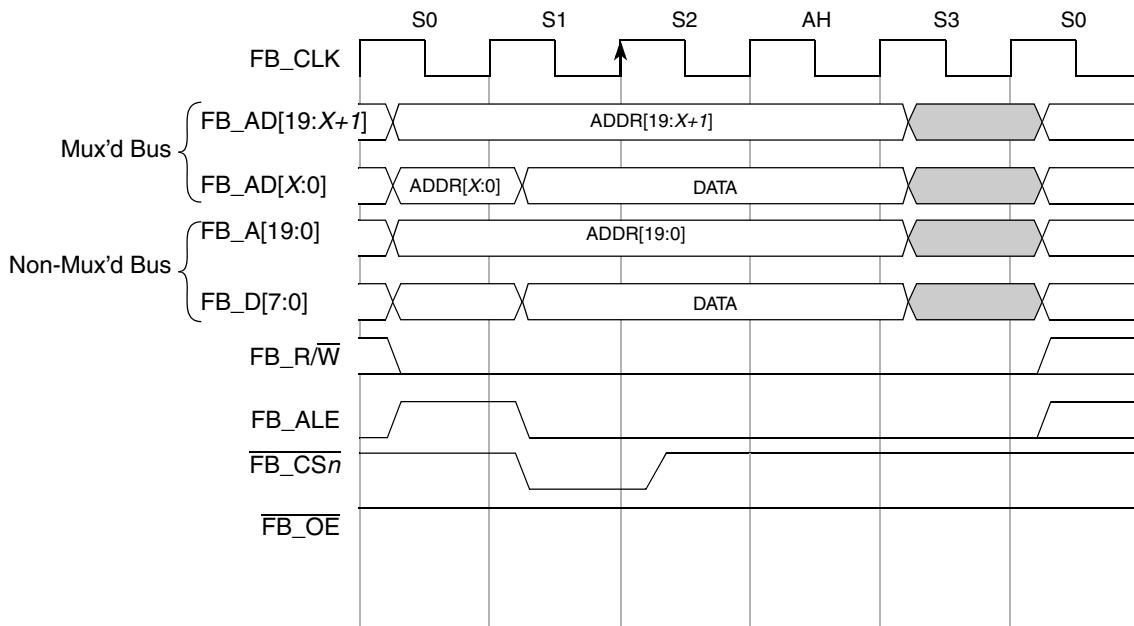
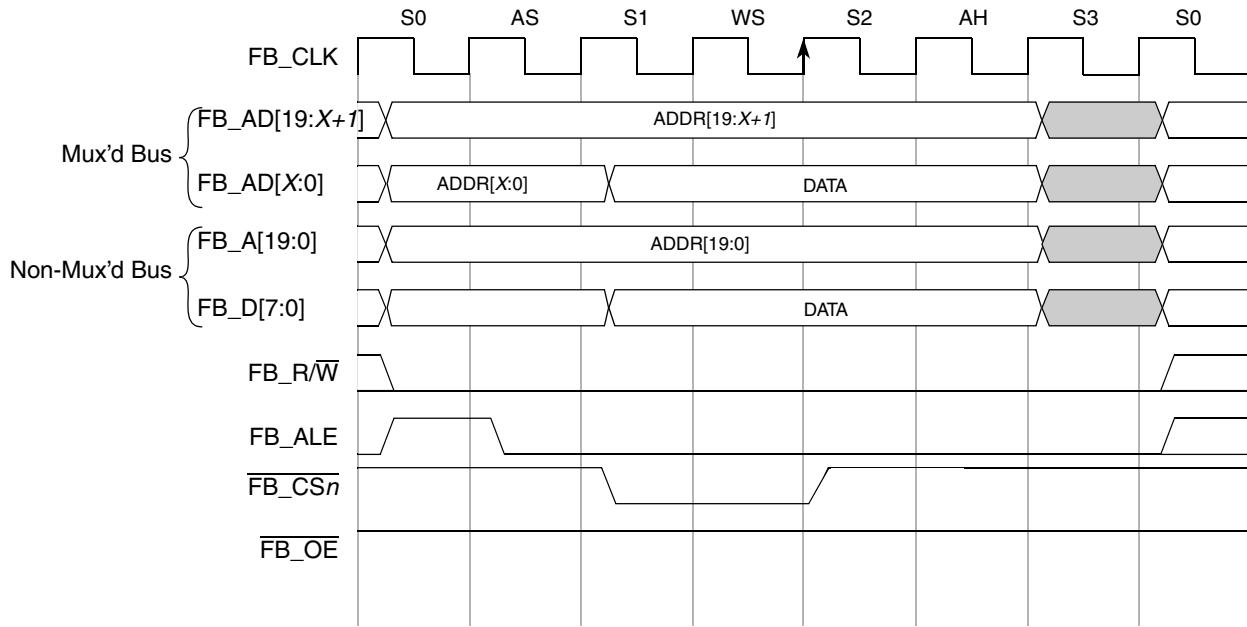


Figure 11-21. Write Cycle with Two-Clock Address Hold (No Wait States)

Figure 11-22 shows a bus cycle using address setup, wait states, and address hold.



**Figure 11-22. Write Cycle with Two-Clock Address Setup and Two-Clock Hold (One Wait State)**

### 11.4.7 Bus Errors

There are certain accesses to the Mini-FlexBus that cause the system bus to hang. It is important to have a good access-error handler to manage these conditions.

One such access is if  $\text{CSCR}_n[\text{AA}]$  is cleared, the system hangs. Four other types of accesses cause the access to terminate with a bus error.

- Mini-Flexbus module disabled using the platform peripheral power management control. Mini-FlexBus accesses cause an error termination on the bus and prohibit the access to the Mini-FlexBus.
- Attempted writes to space defined as write protected ( $\text{CSMR}_n[\text{WP}]$  is set) are terminated with an error response and the access is inhibited to the Mini-FlexBus.
- Mini-FlexBus access not hitting in either chip select region is terminated with an error response and the access is inhibited to the Mini-FlexBus.
- Mini-FlexBus access hitting in both chip select regions is terminated with an error response and the access is inhibited to the Mini-FlexBus

# Chapter 12

## Real-Time Counter (RTC)

### 12.1 Introduction

The Real-Time Counter (RTC) module consists of one 8-bit counter, one 8-bit comparator, several binary-based and decimal-based prescaler dividers, two clock sources, and one programmable periodic interrupt. This module can be used for time-of-day, calendar or any task scheduling functions. It can also serve as a cyclic wakeup from low power modes without the need of external components.

#### NOTE

- For details on low-power mode operation, refer to [Table 3-5](#) in [Chapter 3, “Modes of Operation”](#).
- Use pin mux control registers from [Section 2.3, “Pin Mux Controls”](#) to assign RTC signals to the MCF51CN128 package pins.
- Most pin functions default to GPIO and must be software configured before using RTC.

## 12.1.1 Features

Features of the RTC module include:

- 8-bit up-counter
  - 8-bit modulo match limit
  - Software controllable periodic interrupt on match
- Three software selectable clock sources for input to prescaler with selectable binary-based and decimal-based divider values
  - 1-kHz internal low-power oscillator (LPO)
  - External clock (ERCLK)
  - 32-kHz internal clock (IRCLK)

## 12.1.2 Modes of Operation

This section defines the operation in stop, wait and background debug modes.

### 12.1.2.1 Wait Mode

The RTC continues to run in wait mode if enabled before executing the appropriate instruction. Therefore, the RTC can bring the MCU out of wait mode if the real-time interrupt is enabled. For lowest possible current consumption, the RTC should be stopped by software if not needed as an interrupt source during wait mode.

### 12.1.2.2 Stop Modes

The RTC continues to run in stop2 or stop3 mode if the RTC is enabled before executing the STOP instruction. Therefore, the RTC can bring the MCU out of stop modes with no external components, if the real-time interrupt is enabled.

The LPO clock can be used in stop2 and stop3 modes. ERCLK and IRCLK clocks are only available in stop3 mode.

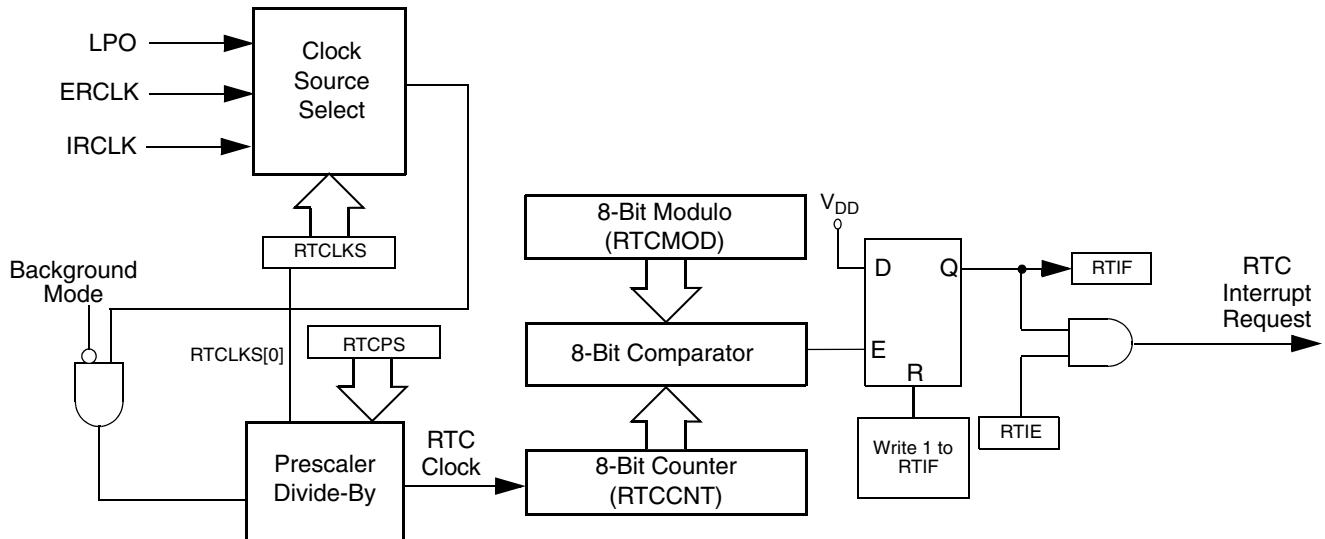
Power consumption is lower when all clock sources are disabled, but in that case, the real-time interrupt cannot wakeup the MCU from stop modes.

### 12.1.2.3 Active Background Mode

The RTC suspends all counting during active background mode until the microcontroller returns to normal user operating mode. Counting resumes from the suspended value as long as the RTCMOD register is not written and the RTCPS and RTCLKS bits are not altered.

### 12.1.3 Block Diagram

The block diagram for the RTC module is shown in [Figure 12-1](#).



**Figure 12-1. Real-Time Counter (RTC) Block Diagram**

## 12.2 External Signal Description

The RTC does not include any off-chip signals.

## 12.3 Register Definition

The RTC includes a status and control register, an 8-bit counter register, and an 8-bit modulo register.

Refer to the direct-page register summary in [Chapter 4, “Memory”](#) for the absolute address assignments for all RTC registers. This section refers to registers and control bits only by their names and relative address offsets.

[Table 12-1](#) is a summary of RTC registers.

**Table 12-1. RTC Register Summary**

Name		7	6	5	4	3	2	1	0								
RTCSC	R	RTIF	RTCLKS		RTIE	RTCPs											
	W																
RTCCNT	R	RTCCNT															
	W																
RTCMOD	R	RTCMOD															
	W																

### 12.3.1 RTC Status and Control Register (RTCSC)

RTCSC contains the real-time interrupt status flag (RTIF), the clock select bits (RTCLKS), the real-time interrupt enable bit (RTIE), and the prescaler select bits (RTCPS).

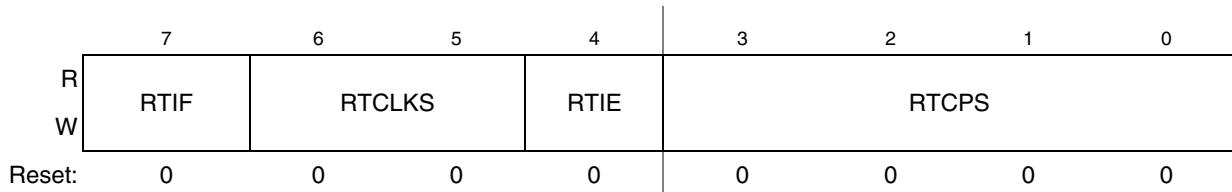


Figure 12-2. RTC Status and Control Register (RTCSC)

Table 12-2. RTCSC Field Descriptions

Field	Description
7 RTIF	<b>Real-Time Interrupt Flag.</b> This status bit indicates the RTC counter register reached the value in the RTC modulo register. Writing a logic 0 has no effect. Writing a logic 1 clears the bit and the real-time interrupt request. Reset clears RTIF. 0 RTC counter has not reached the value in the RTC modulo register. 1 RTC counter has reached the value in the RTC modulo register.
6–5 RTCLKS	<b>Real-Time Clock Source Select.</b> These two read/write bits select the clock source input to the RTC prescaler. Changing the clock source clears the prescaler and RTCCNT counters. When selecting a clock source, ensure that the clock source is properly enabled (if applicable) to ensure correct operation of the RTC. Reset clears RTCLKS. 00 Real-time clock source is the 1-kHz low power oscillator (LPO) 01 Real-time clock source is the external clock (ERCLK) 1x Real-time clock source is the internal clock (IRCLK)
4 RTIE	<b>Real-Time Interrupt Enable.</b> This read/write bit enables real-time interrupts. If RTIE is set, then an interrupt is generated when RTIF is set. Reset clears RTIE. 0 Real-time interrupt requests are disabled. Use software polling. 1 Real-time interrupt requests are enabled.
3–0 RTCPS	<b>Real-Time Clock Prescaler Select.</b> These four read/write bits select binary-based or decimal-based divide-by values for the clock source. See Table 12-3. Changing the prescaler value clears the prescaler and RTCCNT counters. Reset clears RTCPS.

Table 12-3. RTC Prescaler Divide-by values

RTCLKS[0]	RTCPS															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	Off	$2^3$	$2^5$	$2^6$	$2^7$	$2^8$	$2^9$	$2^{10}$	1	2	$2^2$	10	$2^4$	$10^2$	$5 \times 10^2$	$10^3$
1	Off	$2^{10}$	$2^{11}$	$2^{12}$	$2^{13}$	$2^{14}$	$2^{15}$	$2^{16}$	$10^3$	$2 \times 10^3$	$5 \times 10^3$	$10^4$	$2 \times 10^4$	$5 \times 10^4$	$10^5$	$2 \times 10^5$

### 12.3.2 RTC Counter Register (RTCCNT)

RTCCNT is the read-only value of the current RTC count of the 8-bit counter.

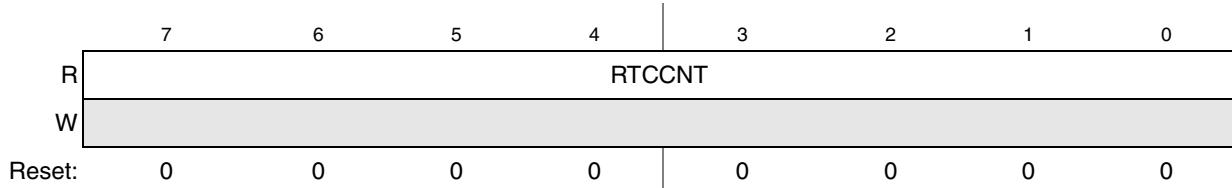


Figure 12-3. RTC Counter Register (RTCCNT)

Table 12-4. RTCCNT Field Descriptions

Field	Description
7:0 RTCCNT	<b>RTC Count.</b> These eight read-only bits contain the current value of the 8-bit counter. Writes have no effect to this register. Reset, writing to RTCMOD, or writing different values to RTCLKS and RTCPS clear the count to 0x00.

### 12.3.3 RTC Modulo Register (RTCMOD)

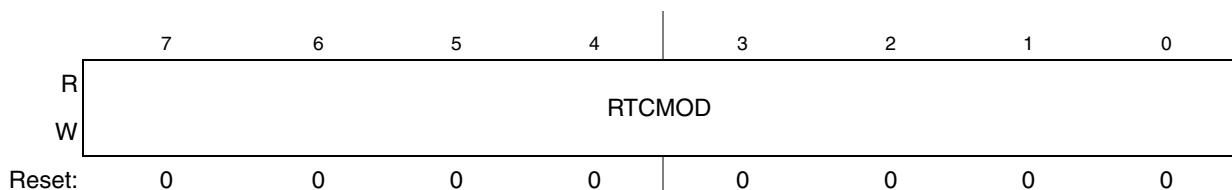


Figure 12-4. RTC Modulo Register (RTCMOD)

Table 12-5. RTCMOD Field Descriptions

Field	Description
7:0 RTCMOD	<b>RTC Modulo.</b> These eight read/write bits contain the modulo value used to reset the count to 0x00 upon a compare match and set the RTIF status bit. A value of 0x00 sets the RTIF bit on each rising edge of the prescaler output. Writing to RTCMOD resets the prescaler and the RTCCNT counters to 0x00. Reset sets the modulo to 0x00.

## 12.4 Functional Description

The RTC is composed of a main 8-bit up-counter with an 8-bit modulo register, a clock source selector, and a prescaler block with binary-based and decimal-based selectable values. The module also contains software selectable interrupt logic.

After any MCU reset, the counter is stopped and reset to 0x00, the modulus register is set to 0x00, and the prescaler is off. The 1-kHz internal oscillator clock is selected as the default clock source. To start the prescaler, write any value other than zero to the prescaler select bits (RTCPS).

Three clock sources are software selectable: the low power oscillator clock (LPO), the external clock (ERCLK), and the internal clock (IRCLK). The RTC clock select bits (RTCLKS) select the desired clock source. If a different value is written to RTCLKS, the prescaler and RTCCNT counters are reset to 0x00.

RTCPS and the RTCLKS[0] bit select the desired divide-by value. If a different value is written to RTCPS, the prescaler and RTCCNT counters are reset to 0x00. [Table 12-6](#) shows different prescaler period values.

**Table 12-6. Prescaler Period**

RTCPS	1-kHz Internal Clock (RTCLKS = 00)	1-MHz External Clock (RTCLKS = 01)	32-kHz Internal Clock (RTCLKS = 10)	32-kHz Internal Clock (RTCLKS = 11)
0000	Off	Off	Off	Off
0001	8 ms	1.024 ms	250 µs	32 ms
0010	32 ms	2.048 ms	1 ms	64 ms
0011	64 ms	4.096 ms	2 ms	128 ms
0100	128 ms	8.192 ms	4 ms	256 ms
0101	256 ms	16.4 ms	8 ms	512 ms
0110	512 ms	32.8 ms	16 ms	1.024 s
0111	1.024 s	65.5 ms	32 ms	2.048 s
1000	1 ms	1 ms	31.25 µs	31.25 ms
1001	2 ms	2 ms	62.5 µs	62.5 ms
1010	4 ms	5 ms	125 µs	156.25 ms
1011	10 ms	10 ms	312.5 µs	312.5 ms
1100	16 ms	20 ms	0.5 ms	0.625 s
1101	0.1 s	50 ms	3.125 ms	1.5625 s
1110	0.5 s	0.1 s	15.625 ms	3.125 s
1111	1 s	0.2 s	31.25 ms	6.25 s

The RTC modulo register (RTCMOD) allows the compare value to be set to any value from 0x00 to 0xFF. When the counter is active, the counter increments at the selected rate until the count matches the modulo value. When these values match, the counter resets to 0x00 and continues counting. The real-time interrupt flag (RTIF) is set when a match occurs. The flag sets on the transition from the modulo value to 0x00. Writing to RTCMOD resets the prescaler and the RTCCNT counters to 0x00.

The RTC allows for an interrupt to be generated when RTIF is set. To enable the real-time interrupt, set the real-time interrupt enable bit (RTIE) in RTCSC. RTIF is cleared by writing a 1 to RTIF.

### 12.4.1 RTC Operation Example

This section shows an example of the RTC operation as the counter reaches a matching value from the modulo register.

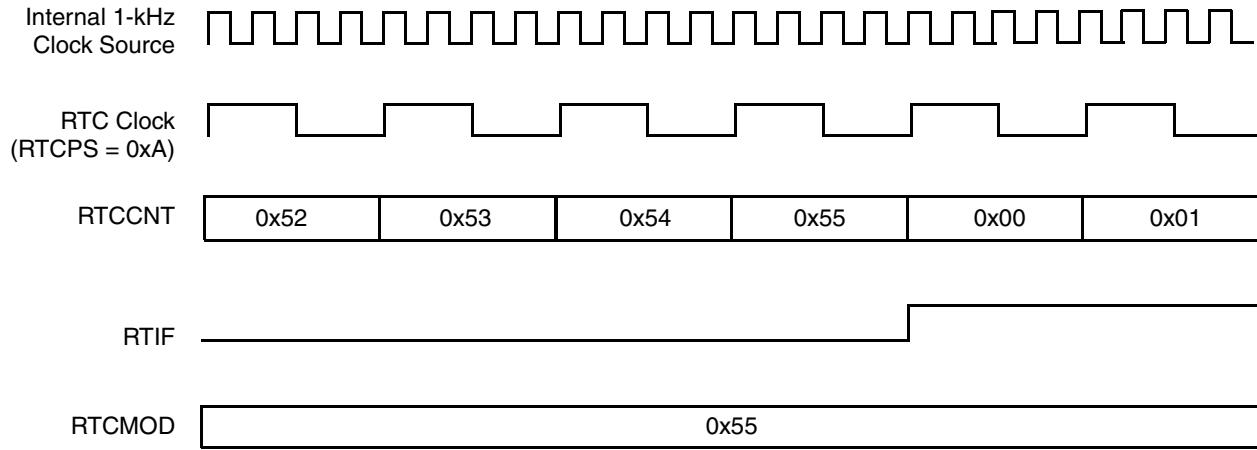


Figure 12-5. RTC Counter Overflow Example

In the example of Figure 12-5, the selected clock source is the 1-kHz internal oscillator clock source. The prescaler (RTCPS) is set to 0xA or divide-by-4. The modulo value in the RTCMOD register is set to 0x55. When the counter, RTCCNT, reaches the modulo value of 0x55, the counter overflows to 0x00 and continues counting. The real-time interrupt flag, RTIF, sets when the counter value changes from 0x55 to 0x00. A real-time interrupt is generated when RTIF is set, if RTIE is set.

## 12.5 Initialization/Application Information

This section provides example code to give you some basic direction on how to initialize and configure the RTC module. The example software is implemented in C language.

The example below shows how to implement time of day with the RTC using the 1-kHz clock source to achieve the lowest possible power consumption. Because the 1-kHz clock source is not as accurate as a crystal, software can be added for any adjustments. For accuracy without adjustments at the expense of additional power consumption, the external clock (ERCLK) or the internal clock (IRCLK) can be selected with appropriate prescaler and modulo values.

```

/* Initialize the elapsed time counters */
Seconds = 0;
Minutes = 0;
Hours = 0;
Days=0;

/* Configure RTC to interrupt every 1 second from 1-kHz clock source */
RTCMOD.byte = 0x00;
RTCSC.byte = 0x1F;

/*****************/
Function Name : RTC_ISR
Notes : Interrupt service routine for RTC module.
/*****************/
#pragma TRAP_PROC
void RTC_ISR(void)
{
    /* Clear the interrupt flag */

```

```
RTCSC.byte = RTCSC.byte | 0x80;
/* RTC interrupts every 1 Second */
Seconds++;
/* 60 seconds in a minute */
if (Seconds > 59){
Minutes++;
Seconds = 0;
}
/* 60 minutes in an hour */
if (Minutes > 59){
Hours++;
Minutes = 0;
}
/* 24 hours in a day */
if (Hours > 23){
Days++;
Hours = 0;
}
}
```

# Chapter 13

## Serial Communication Interface (SCI)

### 13.1 Introduction

The SCI allows asynchronous serial communications with peripheral devices and other CPUs.

#### NOTE

- MCF51CN128 series devices do not include stop1 low-power mode. Ignore references to stop1 in this chapter.
- For details on low-power mode operation, refer to [Table 3-5](#) in [Chapter 3, “Modes of Operation”](#).
- Use pin mux control registers from [Section 2.3, “Pin Mux Controls”](#) to assign SCI signals to the MCF51CN128 package pins.
- Most pin functions default to GPIO and must be software configured before using SCI.

#### 13.1.1 Module Block Diagram

[Figure 13-1](#) shows a block diagram of the SCI module.

## Serial Communication Interface (SCI)

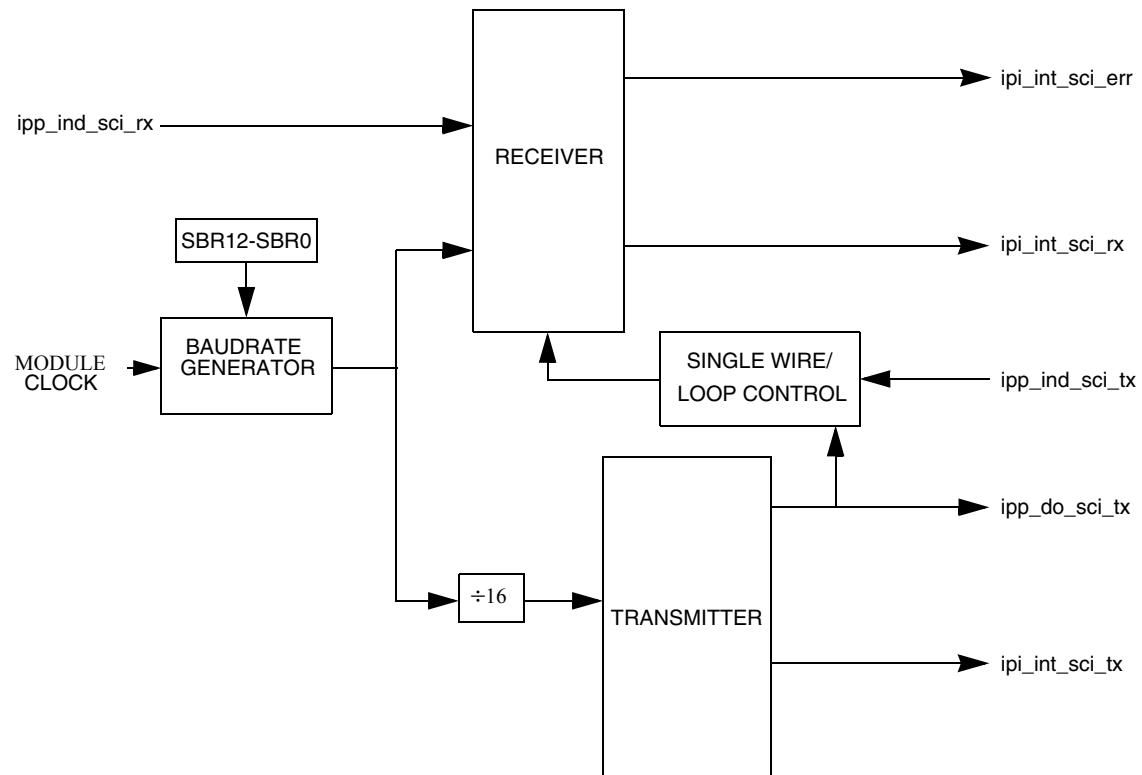


Figure 13-1. SCI Module Block Diagram

### 13.1.2 Features

Features of SCI module include:

- Full-duplex, standard non-return-to-zero (NRZ) format
- Double-buffered transmitter and receiver with separate enables
- Programmable baud rates (13-bit modulo divider)
- Interrupt-driven or polled operation:
  - Transmit data register empty and transmission complete
  - Receive data register full
  - Receive overrun, parity error, framing error, and noise error
  - Idle receiver detect
  - Active edge on receive pin
  - Break detect supporting LIN
- Hardware parity generation and checking
- Programmable 8-bit or 9-bit character length
- Receiver wakeup by idle-line or address-mark
- Optional 13-bit break character generation / 11-bit break character detection
- Selectable transmitter output polarity

### 13.1.3 Modes of Operation

See [Section 13.3, “Functional Description,”](#) for details concerning SCI operation in these modes:

- 8- and 9-bit data modes
- Stop mode operation
- Loop mode
- Single-wire mode

### 13.1.4 Block Diagram

Figure 13-2 shows the transmitter portion of the SCI.

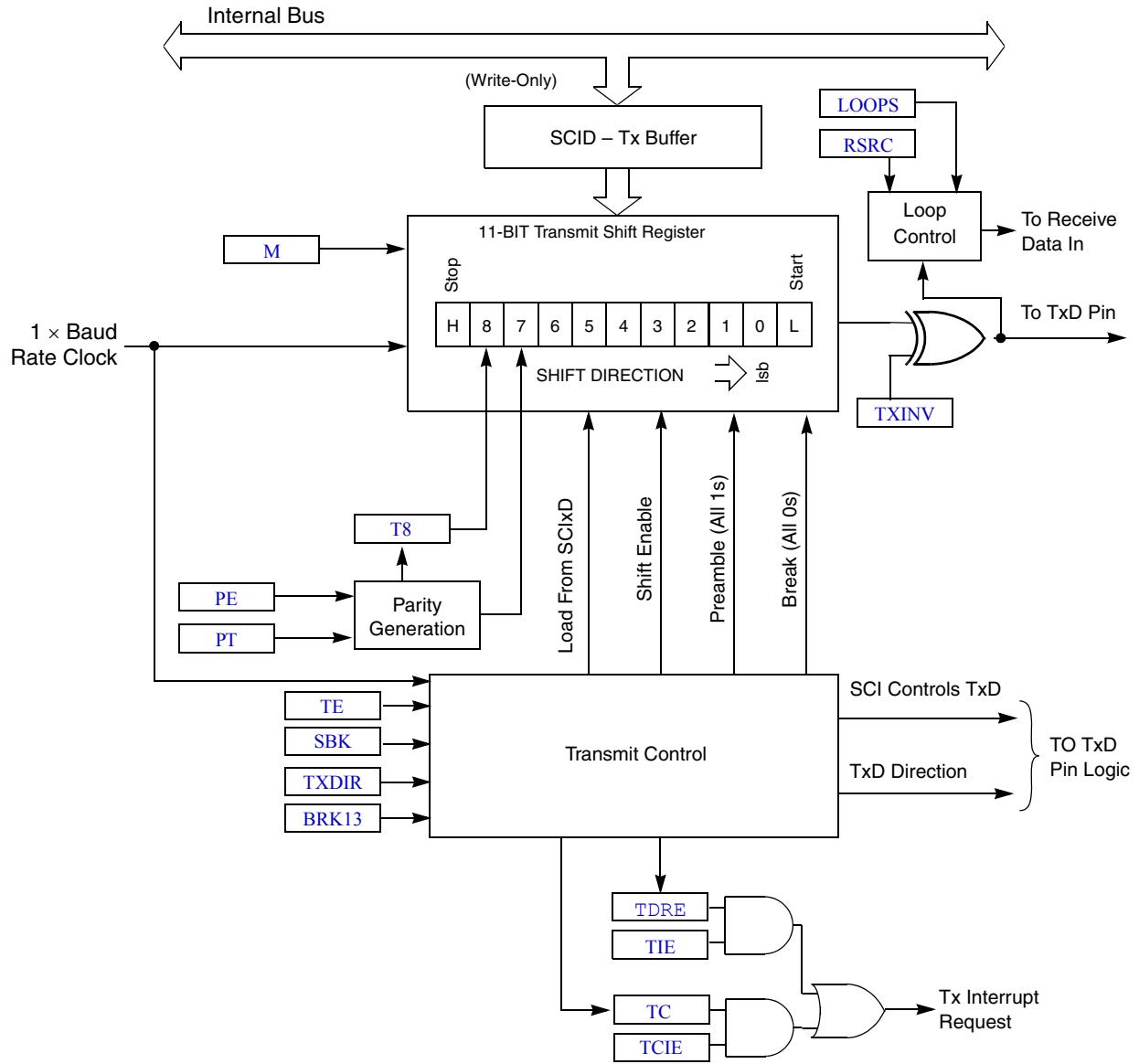
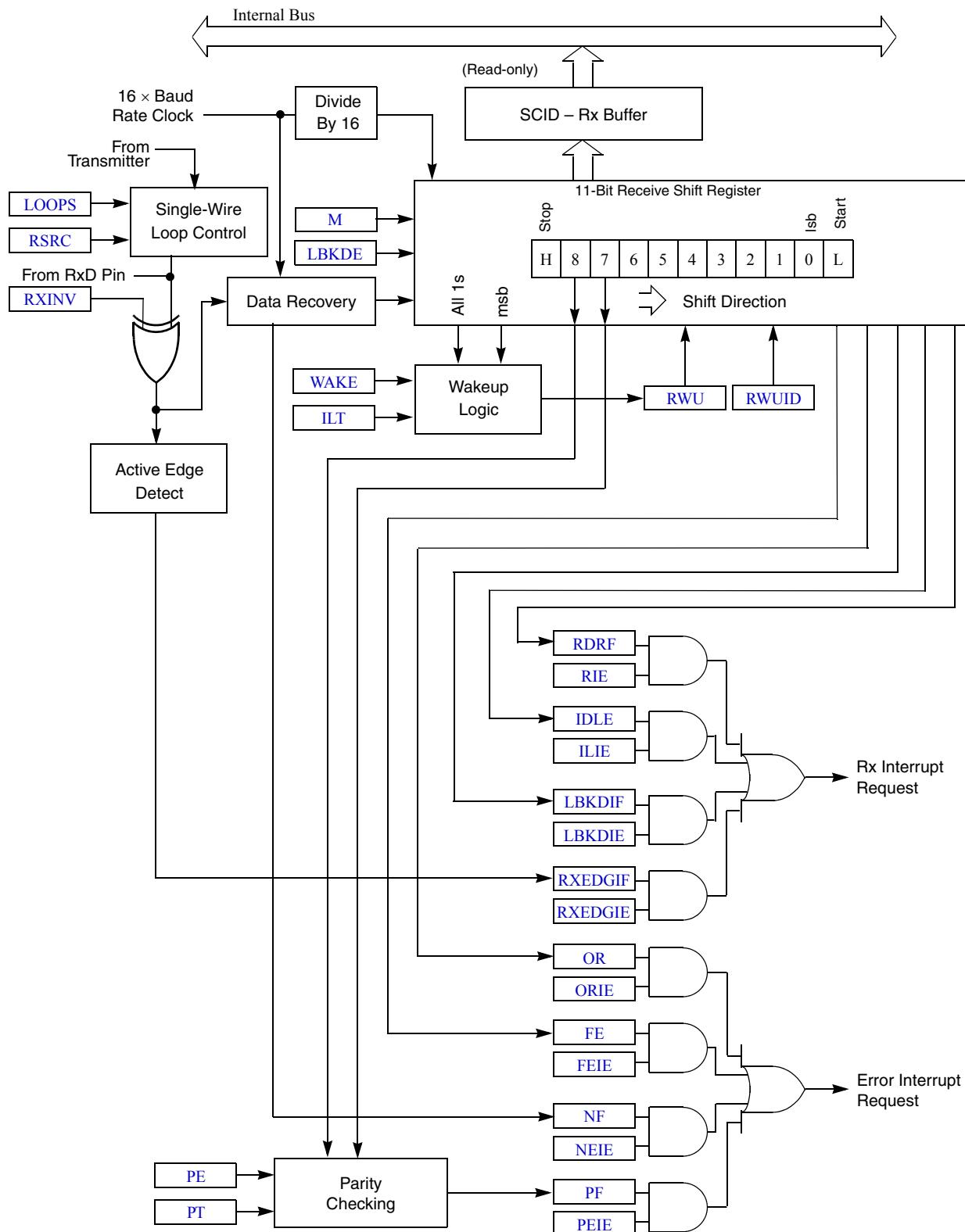


Figure 13-2. SCI Transmitter Block Diagram

Figure 13-3 shows the receiver portion of the SCI.



**Figure 13-3. SCI Receiver Block Diagram**

## 13.2 Register Definition

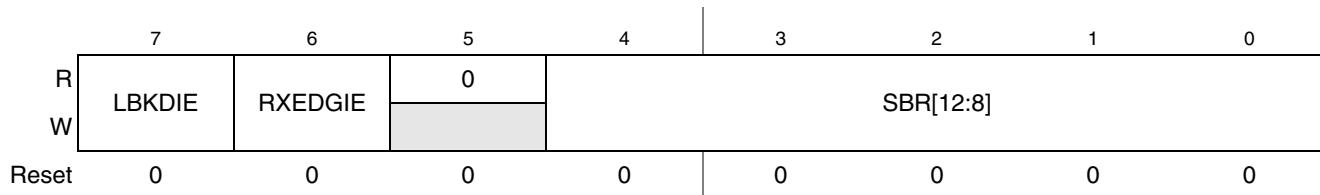
The SCI has eight 8-bit registers to control baud rate, select SCI options, report SCI status, and for transmit/receive data.

Refer to the direct-page register summary in [Chapter 4, “Memory,”](#) or the absolute address assignments for all SCI registers. This section refers to registers and control bits only by their names. A Freescale-provided equate or header file is used to translate these names into the appropriate absolute addresses.

### 13.2.1 SCI Baud Rate Registers (SCIxBDH, SCIxBDL)

This pair of registers controls the prescale divisor for SCI baud rate generation. To update the 13-bit baud rate setting [SBR12:SBR0], first write to SCIxBDH to buffer the high half of the new value and then write to SCIxBDL. The working value in SCIxBDH does not change until SCIxBDL is written.

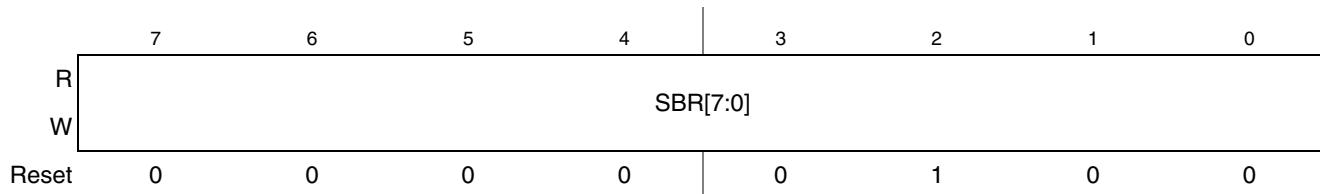
SCIxBDL is reset to a non-zero value, so after reset the baud rate generator remains disabled until the first time the receiver or transmitter is enabled (RE or TE bits in SCIxC2 are written to 1).



**Figure 13-4. SCI Baud Rate Register (SCIxBDH)**

**Table 13-1. SCIxBDH Field Descriptions**

Field	Description
7 LBKDI	LIN Break Detect Interrupt Enable (for LBKDIF) 0 Hardware interrupts from LBKDIF disabled (use polling). 1 Hardware interrupt requested when LBKDIF flag is 1.
6 RXEDGIE	RxD Input Active Edge Interrupt Enable (for RXEDGIF) 0 Hardware interrupts from RXEDGIF disabled (use polling). 1 Hardware interrupt requested when RXEDGIF flag is 1.
4–0 SBR[12:8]	Baud Rate Modulo Divisor. The 13 bits in SBR[12:0] are referred to collectively as BR, and they set the modulo divide rate for the SCI baud rate generator. When BR is cleared, the SCI baud rate generator is disabled to reduce supply current. When BR is 1 – 8191, the SCI baud rate equals SCI module clock/(16 × BR). See also BR bits in <a href="#">Table 13-2</a> .



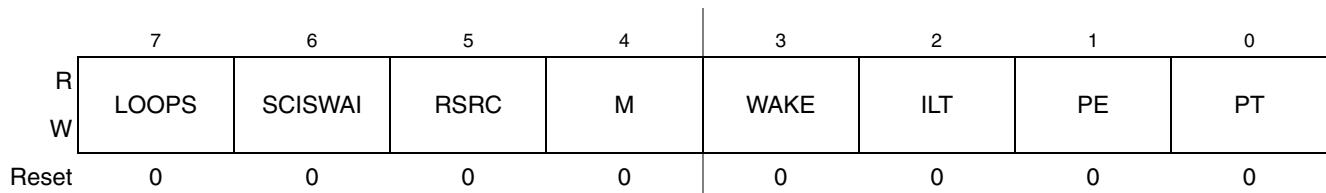
**Figure 13-5. SCI Baud Rate Register (SCIxBDL)**

**Table 13-2. SCIxBDL Field Descriptions**

Field	Description
7–0 SBR[7:0]	Baud Rate Modulo Divisor. These 13 bits in SBR[12:0] are referred to collectively as BR, and they set the modulo divide rate for the SCI baud rate generator. When BR is cleared, the SCI baud rate generator is disabled to reduce supply current. When BR is 1 – 8191, the SCI baud rate equals SCI module clock/(16 × BR). See also BR bits in <a href="#">Table 13-1</a> .

### 13.2.2 SCI Control Register 1 (SCIxC1)

This read/write register controls various optional features of the SCI system.

**Figure 13-6. SCI Control Register 1 (SCIxC1)****Table 13-3. SCIxC1 Field Descriptions**

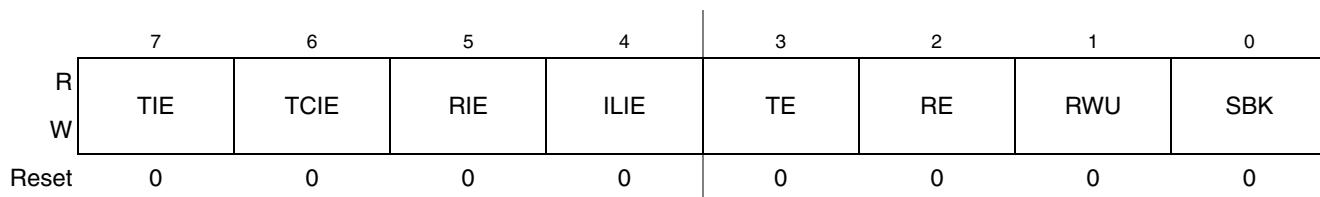
Field	Description
7 LOOPS	Loop Mode Select. Selects between loop back modes and normal 2-pin full-duplex modes. When LOOPS is set, the transmitter output is internally connected to the receiver input. 0 Normal operation — RxD and TxD use separate pins. 1 Loop mode or single-wire mode where transmitter outputs are internally connected to receiver input. (See RSRC bit.) RxD pin is not used by SCI.
6 SCISWAI	SCI Stops in Wait Mode 0 SCI clocks continue to run in wait mode so the SCI can be the source of an interrupt that wakes up the CPU. 1 SCI clocks freeze while CPU is in wait mode.
5 RSRC	Receiver Source Select. This bit has no meaning or effect unless the LOOPS bit is set to 1. When LOOPS is set, the receiver input is internally connected to the TxD pin and RSRC determines whether this connection is also connected to the transmitter output. 0 Provided LOOPS is set, RSRC is cleared, selects internal loop back mode and the SCI does not use the RxD pins. 1 Single-wire SCI mode where the TxD pin is connected to the transmitter output and receiver input.
4 M	9-Bit or 8-Bit Mode Select 0 Normal — start + 8 data bits (lsb first) + stop. 1 Receiver and transmitter use 9-bit data characters start + 8 data bits (lsb first) + 9th data bit + stop.
3 WAKE	Receiver Wakeup Method Select. Refer to <a href="#">Section 13.3.3.2, “Receiver Wakeup Operation”</a> for more information. 0 Idle-line wakeup. 1 Address-mark wakeup.
2 ILT	Idle Line Type Select. Setting this bit to 1 ensures that the stop bit and logic 1 bits at the end of a character do not count toward the 10 or 11 bit times of logic high level needed by the idle line detection logic. Refer to <a href="#">Section 13.3.3.2.1, “Idle-Line Wakeup”</a> for more information. 0 Idle character bit count starts after start bit. 1 Idle character bit count starts after stop bit.

**Table 13-3. SCIx C1 Field Descriptions (continued)**

Field	Description
1 PE	Parity Enable. Enables hardware parity generation and checking. When parity is enabled, the most significant bit (msb) of the data character (eighth or ninth data bit) is treated as the parity bit. 0 No hardware parity generation or checking. 1 Parity enabled.
0 PT	Parity Type. Provided parity is enabled (PE = 1), this bit selects even or odd parity. Odd parity means the total number of 1s in the data character, including the parity bit, is odd. Even parity means the total number of 1s in the data character, including the parity bit, is even. 0 Even parity. 1 Odd parity.

### 13.2.3 SCI Control Register 2 (SCIx C2)

This register can be read or written at any time.

**Figure 13-7. SCI Control Register 2 (SCIx C2)****Table 13-4. SCIx C2 Field Descriptions**

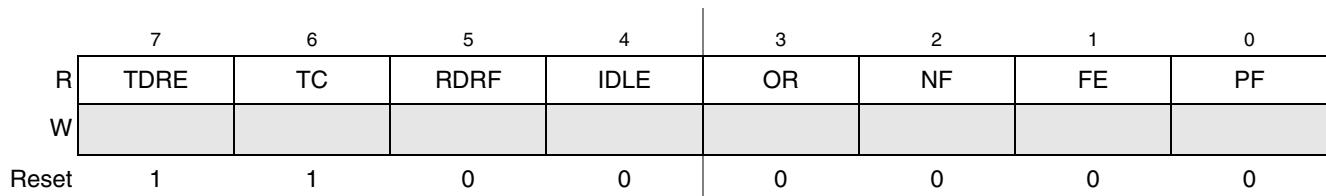
Field	Description
7 TIE	Transmit Interrupt Enable (for TDRE) 0 Hardware interrupts from TDRE disabled (use polling). 1 Hardware interrupt requested when TDRE flag is 1.
6 TCIE	Transmission Complete Interrupt Enable (for TC) 0 Hardware interrupts from TC disabled (use polling). 1 Hardware interrupt requested when TC flag is 1.
5 RIE	Receiver Interrupt Enable (for RDRF) 0 Hardware interrupts from RDRF disabled (use polling). 1 Hardware interrupt requested when RDRF flag is 1.
4 ILIE	Idle Line Interrupt Enable (for IDLE) 0 Hardware interrupts from IDLE disabled (use polling). 1 Hardware interrupt requested when IDLE flag is 1.

**Table 13-4. SCIxC2 Field Descriptions (continued)**

Field	Description
3 TE	Transmitter Enable 0 Transmitter off. 1 Transmitter on.  TE must be 1 to use the SCI transmitter. When TE is set, the SCI forces the TxD pin to act as an output for the SCI system.  When the SCI is configured for single-wire operation (LOOPS = RSRC = 1), TXDIR controls the direction of traffic on the single SCI communication line (TxD pin).  TE can also queue an idle character by clearing TE then setting TE while a transmission is in progress. Refer to <a href="#">Section 13.3.2.1, “Send Break and Queued Idle”</a> for more details.  When TE is written to 0, the transmitter keeps control of the port TxD pin until any data, queued idle, or queued break character finishes transmitting before allowing the pin to revert to a general-purpose I/O pin.
2 RE	Receiver Enable. When the SCI receiver is off, the RxD pin reverts to being a general-purpose port I/O pin. If LOOPS is set the RxD pin reverts to being a general-purpose I/O pin even if RE is set. 0 Receiver off. 1 Receiver on.
1 RWU	Receiver Wakeup Control. This bit can be written to 1 to place the SCI receiver in a standby state where it waits for automatic hardware detection of a selected wakeup condition. The wakeup condition is an idle line between messages (WAKE = 0, idle-line wakeup) or a logic 1 in the most significant data bit in a character (WAKE = 1, address-mark wakeup). Application software sets RWU and (normally) a selected hardware condition automatically clears RWU. Refer to <a href="#">Section 13.3.2.2, “Receiver Wakeup Operation,”</a> for more details. 0 Normal SCI receiver operation. 1 SCI receiver in standby waiting for wakeup condition.
0 SBK	Send Break. Writing a 1 and then a 0 to SBK queues a break character in the transmit data stream. Additional break characters of 10 or 11 (13 or 14 if BRK13 = 1) bit times of logic 0 are queued as long as SBK is set. Depending on the timing of the set and clear of SBK relative to the information currently being transmitted, a second break character may be queued before software clears SBK. Refer to <a href="#">Section 13.3.2.1, “Send Break and Queued Idle”</a> for more details. 0 Normal transmitter operation. 1 Queue break character(s) to be sent.

### 13.2.4 SCI Status Register 1 (SCIxS1)

This register has eight read-only status flags. Writes have no effect. Special software sequences (which do not involve writing to this register) clear these status flags.

**Figure 13-8. SCI Status Register 1 (SCIxS1)**

**Table 13-5. SCIxS1 Field Descriptions**

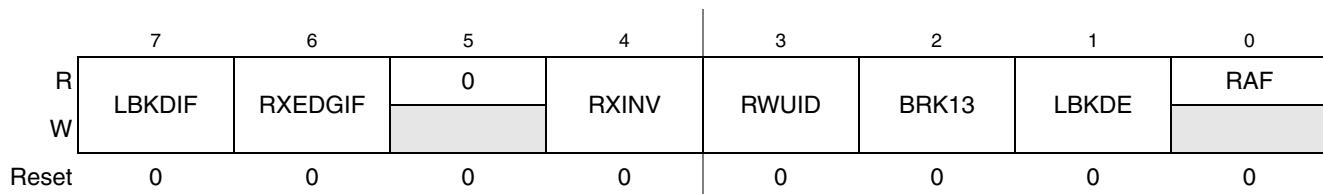
<b>Field</b>	<b>Description</b>
7 TDRE	Transmit Data Register Empty Flag. TDRE is set out of reset and when a transmit data value transfers from the transmit data buffer to the transmit shifter, leaving room for a new character in the buffer. To clear TDRE, read SCIxS1 with TDRE set and then write to the SCI data register (SCIxD). 0 Transmit data register (buffer) full. 1 Transmit data register (buffer) empty.
6 TC	Transmission Complete Flag. TC is set out of reset and when TDRE is set and no data, preamble, or break character is being transmitted. 0 Transmitter active (sending data, a preamble, or a break). 1 Transmitter idle (transmission activity complete). TC is cleared automatically by reading SCIxS1 with TC set and then doing one of the following: <ul style="list-style-type: none"> <li>• Write to the SCI data register (SCIxD) to transmit new data</li> <li>• Queue a preamble by changing TE from 0 to 1</li> <li>• Queue a break character by writing 1 to SBK in SCIx2</li> </ul>
5 RDRF	Receive Data Register Full Flag. RDRF becomes set when a character transfers from the receive shifter into the receive data register (SCIxD). To clear RDRF, read SCIxS1 with RDRF set and then read the SCI data register (SCIxD). 0 Receive data register empty. 1 Receive data register full.
4 IDLE	Idle Line Flag. IDLE is set when the SCI receive line becomes idle for a full character time after a period of activity. When ILT is cleared, the receiver starts counting idle bit times after the start bit. If the receive character is all 1s, these bit times and the stop bit time count toward the full character time of logic high (10 or 11 bit times depending on the M control bit) needed for the receiver to detect an idle line. When ILT is set, the receiver doesn't start counting idle bit times until after the stop bit. The stop bit and any logic high bit times at the end of the previous character do not count toward the full character time of logic high needed for the receiver to detect an idle line. To clear IDLE, read SCIxS1 with IDLE set and then read the SCI data register (SCIxD). After IDLE has been cleared, it cannot become set again until after a new character has been received and RDRF has been set. IDLE is set only once even if the receive line remains idle for an extended period. 0 No idle line detected. 1 Idle line was detected.
3 OR	Receiver Overrun Flag. OR is set when a new serial character is ready to be transferred to the receive data register (buffer), but the previously received character has not been read from SCIxD yet. In this case, the new character (and all associated error information) is lost because there is no room to move it into SCIxD. To clear OR, read SCIxS1 with OR set and then read the SCI data register (SCIxD). 0 No overrun. 1 Receive overrun (new SCI data lost).
2 NF	Noise Flag. The advanced sampling technique used in the receiver takes seven samples during the start bit and three samples in each data bit and the stop bit. If any of these samples disagrees with the rest of the samples within any bit time in the frame, the flag NF is set at the same time as RDRF is set for the character. To clear NF, read SCIxS1 and then read the SCI data register (SCIxD). 0 No noise detected. 1 Noise detected in the received character in SCIxD.

**Table 13-5. SCIxS1 Field Descriptions (continued)**

Field	Description
1 FE	Framing Error Flag. FE is set at the same time as RDRF when the receiver detects a logic 0 where the stop bit was expected. This suggests the receiver was not properly aligned to a character frame. To clear FE, read SCIxS1 with FE set and then read the SCI data register (SCIxD). 0 No framing error detected. This does not guarantee the framing is correct. 1 Framing error.
0 PF	Parity Error Flag. PF is set at the same time as RDRF when parity is enabled (PE = 1) and the parity bit in the received character does not agree with the expected parity value. To clear PF, read SCIxS1 and then read the SCI data register (SCIxD). 0 No parity error. 1 Parity error.

### 13.2.5 SCI Status Register 2 (SCIxS2)

This register contains one read-only status flag.

**Figure 13-9. SCI Status Register 2 (SCIxS2)****Table 13-6. SCIxS2 Field Descriptions**

Field	Description
7 LBKDIF	LIN Break Detect Interrupt Flag. LBKDIF is set when the LIN break detect circuitry is enabled and a LIN break character is detected. LBKDIF is cleared by writing a 1 to it. 0 No LIN break character has been detected. 1 LIN break character has been detected.
6 RXEDGIF	RxD Pin Active Edge Interrupt Flag. RXEDGIF is set when an active edge (falling if RXINV = 0, rising if RXINV=1) on the RxD pin occurs. RXEDGIF is cleared by writing a 1 to it. 0 No active edge on the receive pin has occurred. 1 An active edge on the receive pin has occurred.
4 RXINV <sup>1</sup>	Receive Data Inversion. Setting this bit reverses the polarity of the received data input. 0 Receive data not inverted 1 Receive data inverted
3 RWUID	Receive Wake Up Idle Detect. RWUID controls whether the idle character that wakes up the receiver sets the IDLE bit. 0 During receive standby state (RWU = 1), the IDLE bit does not get set upon detection of an idle character. 1 During receive standby state (RWU = 1), the IDLE bit gets set upon detection of an idle character.
2 BRK13	Break Character Generation Length. BRK13 selects a longer transmitted break character length. Detection of a framing error is not affected by the state of this bit. 0 Break character is transmitted with length of 10 bit times (11 if M = 1) 1 Break character is transmitted with length of 13 bit times (14 if M = 1)

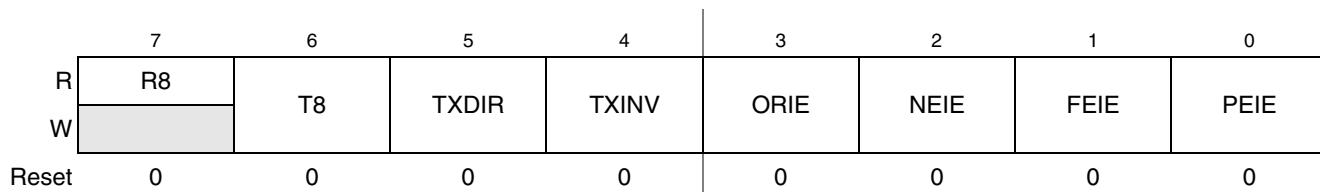
**Table 13-6. SCIxS2 Field Descriptions (continued)**

Field	Description
1 LBKDE	LIN Break Detection Enable. LBKDE selects a longer break character detection length. While LBKDE is set, framing error (FE) and receive data register full (RDRF) flags are prevented from setting. 0 Break character is detected at length of 10 bit times (11 if M = 1). 1 Break character is detected at length of 11 bit times (12 if M = 1).
0 RAF	Receiver Active Flag. RAF is set when the SCI receiver detects the beginning of a valid start bit, and RAF is cleared automatically when the receiver detects an idle line. This status flag can be used to check whether an SCI character is being received before instructing the MCU to go to stop mode. 0 SCI receiver idle waiting for a start bit. 1 SCI receiver active (RxD input not idle).

<sup>1</sup> Setting RXINV inverts the RxD input for all cases: data bits, start and stop bits, break, and idle.

When using an internal oscillator in a LIN system, it is necessary to raise the break detection threshold one bit time. Under the worst case timing conditions allowed in LIN, it is possible that a 0x00 data character can appear to be 10.26 bit times long at a slave running 14% faster than the master. This would trigger normal break detection circuitry designed to detect a 10-bit break symbol. When the LBKDE bit is set, framing errors are inhibited and the break detection threshold changes from 10 bits to 11 bits, preventing false detection of a 0x00 data character as a LIN break symbol.

### 13.2.6 SCI Control Register 3 (SCIxC3)

**Figure 13-10. SCI Control Register 3 (SCIxC3)****Table 13-7. SCIxC3 Field Descriptions**

Field	Description
7 R8	Ninth Data Bit for Receiver. When the SCI is configured for 9-bit data (M = 1), R8 can be thought of as a ninth receive data bit to the left of the msb of the buffered data in the SCIxD register. When reading 9-bit data, read R8 before reading SCIxD because reading SCIxD completes automatic flag clearing sequences that could allow R8 and SCIxD to be overwritten with new data.
6 T8	Ninth Data Bit for Transmitter. When the SCI is configured for 9-bit data (M = 1), T8 may be thought of as a ninth transmit data bit to the left of the msb of the data in the SCIxD register. When writing 9-bit data, the entire 9-bit value is transferred to the SCI shift register after SCIxD is written so T8 should be written (if it needs to change from its previous value) before SCIxD is written. If T8 does not need to change in the new value (such as when it is used to generate mark or space parity), it need not be written each time SCIxD is written.
5 TXDIR	TxD Pin Direction in Single-Wire Mode. When the SCI is configured for single-wire half-duplex operation (LOOPS = RSRC = 1), this bit determines the direction of data at the TxD pin. 0 TxD pin is an input in single-wire mode. 1 TxD pin is an output in single-wire mode.

**Table 13-7. SCIx C3 Field Descriptions (continued)**

Field	Description
4 TXINV <sup>1</sup>	Transmit Data Inversion. Setting this bit reverses the polarity of the transmitted data output. 0 Transmit data not inverted 1 Transmit data inverted
3 ORIE	Overrun Interrupt Enable. This bit enables the overrun flag (OR) to generate hardware interrupt requests. 0 OR interrupts disabled (use polling). 1 Hardware interrupt requested when OR is set.
2 NEIE	Noise Error Interrupt Enable. This bit enables the noise flag (NF) to generate hardware interrupt requests. 0 NF interrupts disabled (use polling). 1 Hardware interrupt requested when NF is set.
1 FEIE	Framing Error Interrupt Enable. This bit enables the framing error flag (FE) to generate hardware interrupt requests. 0 FE interrupts disabled (use polling). 1 Hardware interrupt requested when FE is set.
0 PEIE	Parity Error Interrupt Enable. This bit enables the parity error flag (PF) to generate hardware interrupt requests. 0 PF interrupts disabled (use polling). 1 Hardware interrupt requested when PF is set.

<sup>1</sup> Setting TXINV inverts the TxD output for all cases: data bits, start and stop bits, break, and idle.

### 13.2.7 SCI Data Register (SCIx D)

This register is actually two separate registers. Reads return the contents of the read-only receive data buffer and writes go to the write-only transmit data buffer. Reads and writes of this register are also involved in the automatic flag clearing mechanisms for the SCI status flags.

	7	6	5	4		3	2	1	0
R	R7	R6	R5	R4	R3	R2	R1	R0	
W	T7	T6	T5	T4	T3	T2	T1	T0	
Reset	0	0	0	0	0	0	0	0	0

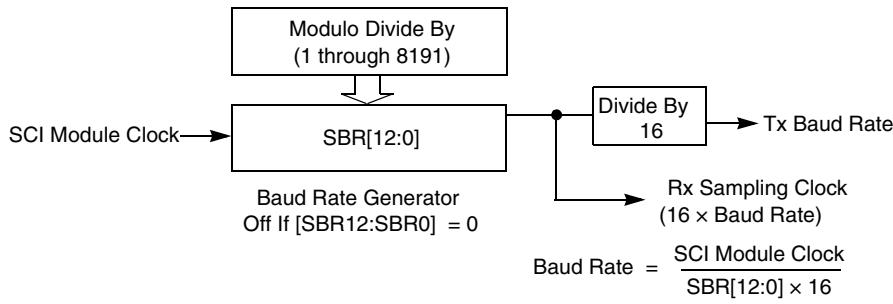
**Figure 13-11. SCI Data Register (SCIx D)**

## 13.3 Functional Description

The SCI allows full-duplex, asynchronous, NRZ serial communication among the MCU and remote devices, including other MCUs. The SCI comprises a baud rate generator, transmitter, and receiver block. The transmitter and receiver operate independently, although they use the same baud rate generator. During normal operation, the MCU monitors the status of the SCI, writes the data to be transmitted, and processes received data. The following describes each of the blocks of the SCI.

### 13.3.1 Baud Rate Generation

As shown in [Figure 13-12](#), the clock source for the SCI baud rate generator is the SCI module clock.

**Figure 13-12. SCI Baud Rate Generation**

SCI communications require the transmitter and receiver (which typically derive baud rates from independent clock sources) to use the same baud rate. Allowed tolerance on this baud frequency depends on the details of how the receiver synchronizes to the leading edge of the start bit and how bit sampling is performed.

The MCU resynchronizes to bit boundaries on every high-to-low transition. In the worst case, there are no such transitions in the full 10- or 11-bit time character frame so any mismatch in baud rate is accumulated for the whole character time. For a Freescale Semiconductor SCI system whose bus frequency is driven by a crystal, the allowed baud rate mismatch is about  $\pm 4.5$  percent for 8-bit data format and about  $\pm 4$  percent for 9-bit data format. Although baud rate modulo divider settings do not always produce baud rates that exactly match standard rates, it is normally possible to get within a few percent, which is acceptable for reliable communications.

### 13.3.2 Transmitter Functional Description

This section describes the overall block diagram for the SCI transmitter, as well as specialized functions for sending break and idle characters. The transmitter block diagram is shown in [Figure 13-2](#).

The transmitter output (Tx<sub>D</sub>) idle state defaults to logic high (TXINV is cleared following reset). The transmitter output is inverted by setting TXINV. The transmitter is enabled by setting the TE bit in SCIx<sub>C2</sub>. This queues a preamble character that is one full character frame of the idle state. The transmitter then remains idle until data is available in the transmit data buffer. Programs store data into the transmit data buffer by writing to the SCI data register (SCIx<sub>D</sub>).

The central element of the SCI transmitter is the transmit shift register that is 10 or 11 bits long depending on the setting in the M control bit. For the remainder of this section, assume M is cleared, selecting the normal 8-bit data mode. In 8-bit data mode, the shift register holds a start bit, eight data bits, and a stop bit. When the transmit shift register is available for a new SCI character, the value waiting in the transmit data register is transferred to the shift register (synchronized with the baud rate clock) and the transmit data register empty (TDRE) status flag is set to indicate another character may be written to the transmit data buffer at SCIx<sub>D</sub>.

If no new character is waiting in the transmit data buffer after a stop bit is shifted out the Tx<sub>D</sub> pin, the transmitter sets the transmit complete flag and enters an idle mode, with Tx<sub>D</sub> high, waiting for more characters to transmit.

Writing 0 to TE does not immediately release the pin to be a general-purpose I/O pin. Any transmit activity in progress must first be completed. This includes data characters in progress, queued idle characters, and queued break characters.

### 13.3.2.1 Send Break and Queued Idle

The SBK control bit in SCIxC2 sends break characters originally used to gain the attention of old teletype receivers. Break characters are a full character time of logic 0 (10 bit times including the start and stop bits). A longer break of 13 bit times can be enabled by setting BRK13. Normally, a program would wait for TDRE to become set to indicate the last character of a message has moved to the transmit shifter, then write 1 and then write 0 to the SBK bit. This action queues a break character to be sent as soon as the shifter is available. If SBK remains 1 when the queued break moves into the shifter (synchronized to the baud rate clock), an additional break character is queued. If the receiving device is another Freescale Semiconductor SCI, the break characters are received as 0s in all eight data bits and a framing error (FE = 1) occurs.

When idle-line wakeup is used, a full character time of idle (logic 1) is needed between messages to wake up any sleeping receivers. Normally, a program would wait for TDRE to become set to indicate the last character of a message has moved to the transmit shifter, then write 0 and then write 1 to the TE bit. This action queues an idle character to be sent as soon as the shifter is available. As long as the character in the shifter does not finish while TE is cleared, the SCI transmitter never actually releases control of the TxD pin. If there is a possibility of the shifter finishing while TE is cleared, set the general-purpose I/O controls so the pin shared with TxD is an output driving a logic 1. This ensures that the TxD line looks like a normal idle line even if the SCI loses control of the port pin between writing 0 and then 1 to TE.

The length of the break character is affected by the BRK13 and M bits as shown below.

**Table 13-8. Break Character Length**

BRK13	M	Break Character Length
0	0	10 bit times
0	1	11 bit times
1	0	13 bit times
1	1	14 bit times

### 13.3.3 Receiver Functional Description

In this section, the receiver block diagram (Figure 13-3) is a guide for the overall receiver functional description. Next, the data sampling technique used to reconstruct receiver data is described in more detail. Finally, two variations of the receiver wakeup function are explained.

The receiver input is inverted by setting RXINV. The receiver is enabled by setting the RE bit in SCIxC2. Character frames consist of a start bit of logic 0, eight (or nine) data bits (lsb first), and a stop bit of logic 1. For information about 9-bit data mode, refer to [Section •, “8- and 9-bit data modes”](#). For the remainder of this discussion, assume the SCI is configured for normal 8-bit data mode.

After receiving the stop bit into the receive shifter, and provided the receive data register is not already full, the data character is transferred to the receive data register and the receive data register full (RDRF)

status flag is set. If RDRF was already set indicating the receive data register (buffer) was already full, the overrun (OR) status flag is set and the new data is lost. Because the SCI receiver is double-buffered, the program has one full character time after RDRF is set before the data in the receive data buffer must be read to avoid a receiver overrun.

When a program detects that the receive data register is full ( $RDRF = 1$ ), it gets the data from the receive data register by reading SCIx.D. The RDRF flag is cleared automatically by a two-step sequence normally satisfied in the course of the user's program that manages receive data. Refer to [Section 13.3.4, "Interrupts and Status Flags,"](#) for more details about flag clearing.

### 13.3.3.1 Data Sampling Technique

The SCI receiver uses a  $16\times$  baud rate clock for sampling. The receiver starts by taking logic level samples at 16 times the baud rate to search for a falling edge on the Rx.D serial data input pin. A falling edge is defined as a logic 0 sample after three consecutive logic 1 samples. The  $16\times$  baud rate clock divides the bit time into 16 segments labeled RT1 through RT16. When a falling edge is located, three more samples are taken at RT3, RT5, and RT7 to make sure this was a real start bit and not merely noise. If at least two of these three samples are 0, the receiver assumes it is synchronized to a receive character.

The receiver then samples each bit time, including the start and stop bits, at RT8, RT9, and RT10 to determine the logic level for that bit. The logic level is interpreted to be that of the majority of the samples taken during the bit time. In the case of the start bit, the bit is assumed to be 0 if at least two of the samples at RT3, RT5, and RT7 are 0 even if one or all of the samples taken at RT8, RT9, and RT10 are 1s. If any sample in any bit time (including the start and stop bits) in a character frame fails to agree with the logic level for that bit, the noise flag (NF) is set when the received character is transferred to the receive data buffer.

The falling edge detection logic continuously looks for falling edges. If an edge is detected, the sample clock is resynchronized to bit times. This improves the reliability of the receiver in the presence of noise or mismatched baud rates. It does not improve worst case analysis because some characters do not have any extra falling edges anywhere in the character frame.

In the case of a framing error, provided the received character was not a break character, the sampling logic that searches for a falling edge is filled with three logic 1 samples so that a new start bit can be detected almost immediately.

In the case of a framing error, the receiver is inhibited from receiving any new characters until the framing error flag is cleared. The receive shift register continues to function, but a complete character cannot transfer to the receive data buffer if FE remains set.

### 13.3.3.2 Receiver Wakeup Operation

Receiver wakeup is a hardware mechanism that allows an SCI receiver to ignore the characters in a message intended for a different SCI receiver. In such a system, all receivers evaluate the first character(s) of each message, and as soon as they determine the message is intended for a different receiver, they write logic 1 to the receiver wake up (RWU) control bit in SCIx.C2. When RWU bit is set, the status flags associated with the receiver (with the exception of the idle bit, IDLE, when RWUID bit is set) are inhibited from setting, thus eliminating the software overhead for handling the unimportant message characters. At

the end of a message, or at the beginning of the next message, all receivers automatically force RWU to 0 so all receivers wake up in time to look at the first character(s) of the next message.

### 13.3.3.2.1 Idle-Line Wakeup

When wake is cleared, the receiver is configured for idle-line wakeup. In this mode, RWU is cleared automatically when the receiver detects a full character time of the idle-line level. The M control bit selects 8-bit or 9-bit data mode that determines how many bit times of idle are needed to constitute a full character time (10 or 11 bit times because of the start and stop bits).

When RWU is one and RWUID is zero, the idle condition that wakes up the receiver does not set the IDLE flag. The receiver wakes up and waits for the first data character of the next message that sets the RDRF flag and generates an interrupt if enabled. When RWUID is one, any idle condition sets the IDLE flag and generates an interrupt if enabled, regardless of whether RWU is zero or one.

The idle-line type (ILT) control bit selects one of two ways to detect an idle line. When ILT is cleared, the idle bit counter starts after the start bit so the stop bit and any logic 1s at the end of a character count toward the full character time of idle. When ILT is set, the idle bit counter does not start until after a stop bit time, so the idle detection is not affected by the data in the last character of the previous message.

### 13.3.3.2.2 Address-Mark Wakeup

When wake is set, the receiver is configured for address-mark wakeup. In this mode, RWU is cleared automatically when the receiver detects a logic 1 in the most significant bit of a received character (eighth bit when M is cleared and ninth bit when M is set).

Address-mark wakeup allows messages to contain idle characters, but requires the msb be reserved for use in address frames. The logic 1 msb of an address frame clears the RWU bit before the stop bit is received and sets the RDRF flag. In this case, the character with the msb set is received even though the receiver was sleeping during most of this character time.

## 13.3.4 Interrupts and Status Flags

The SCI system has three separate interrupt vectors to reduce the amount of software needed to isolate the cause of the interrupt. One interrupt vector is associated with the transmitter for TDRE and TC events. Another interrupt vector is associated with the receiver for RDRF, IDLE, RXEDGIF, and LBKDIF events. A third vector is used for OR, NF, FE, and PF error conditions. Each of these ten interrupt sources can be separately masked by local interrupt enable masks. The flags can be polled by software when the local masks are cleared to disable generation of hardware interrupt requests.

The SCI transmitter has two status flags that can optionally generate hardware interrupt requests. Transmit data register empty (TDRE) indicates when there is room in the transmit data buffer to write another transmit character to SCIXD. If the transmit interrupt enable (TIE) bit is set, a hardware interrupt is requested when TDRE is set. Transmit complete (TC) indicates that the transmitter is finished transmitting all data, preamble, and break characters and is idle with TxD at the inactive level. This flag is often used in systems with modems to determine when it is safe to turn off the modem. If the transmit complete interrupt enable (TCIE) bit is set, a hardware interrupt is requested when TC is set. Instead of hardware

interrupts, software polling may be used to monitor the TDRE and TC status flags if the corresponding TIE or TCIE local interrupt masks are cleared.

When a program detects that the receive data register is full (RDRF = 1), it gets the data from the receive data register by reading SCIxD. The RDRF flag is cleared by reading SCIxS1 while RDRF is set and then reading SCIxD.

When polling is used, this sequence is naturally satisfied in the normal course of the user program. If hardware interrupts are used, SCIxS1 must be read in the interrupt service routine (ISR). Normally, this is done in the ISR anyway to check for receive errors, so the sequence is automatically satisfied.

The IDLE status flag includes logic that prevents it from getting set repeatedly when the RxD line remains idle for an extended period of time. IDLE is cleared by reading SCIxS1 while IDLE is set and then reading SCIxD. After IDLE has been cleared, it cannot become set again until the receiver has received at least one new character and has set RDRF.

If the associated error was detected in the received character that caused RDRF to be set, the error flags — noise flag (NF), framing error (FE), and parity error flag (PF) — are set at the same time as RDRF. These flags are not set in overrun cases.

If RDRF was already set when a new character is ready to be transferred from the receive shifter to the receive data buffer, the overrun (OR) flag is set instead of the data along with any associated NF, FE, or PF condition is lost.

At any time, an active edge on the RxD serial data input pin causes the RXEDGIF flag to set. The RXEDGIF flag is cleared by writing a 1 to it. This function does depend on the receiver being enabled (RE = 1).

### 13.3.5 Additional SCI Functions

The following sections describe additional SCI functions.

#### 13.3.5.1 8- and 9-Bit Data Modes

The SCI system (transmitter and receiver) can be configured to operate in 9-bit data mode by setting the M control bit in SCIxC1. In 9-bit mode, there is a ninth data bit to the left of the msb of the SCI data register. For the transmit data buffer, this bit is stored in T8 in SCIxC3. For the receiver, the ninth bit is held in R8 in SCIxC3.

For coherent writes to the transmit data buffer, write to the T8 bit before writing to SCIxD.

If the bit value to be transmitted as the ninth bit of a new character is the same as for the previous character, it is not necessary to write to T8 again. When data is transferred from the transmit data buffer to the transmit shifter, the value in T8 is copied at the same time data is transferred from SCIxD to the shifter.

The 9-bit data mode is typically used with parity to allow eight bits of data plus the parity in the ninth bit, or it is used with address-mark wakeup so the ninth data bit can serve as the wakeup bit. In custom protocols, the ninth bit can also serve as a software-controlled marker.

### 13.3.5.2 Stop Mode Operation

During all stop modes, clocks to the SCI module are halted.

In stop1 and stop2 modes, all SCI register data is lost and must be re-initialized upon recovery from these two stop modes. No SCI module registers are affected in stop3 mode.

The receive input active edge detect circuit remains active in stop3 mode, but not in stop2. An active edge on the receive input brings the CPU out of stop3 mode if the interrupt is not masked (RXEDGIE = 1).

Because the clocks are halted, the SCI module resumes operation upon exit from stop (only in stop3 mode). Software should ensure stop mode is not entered while there is a character being transmitted out of or received into the SCI module.

### 13.3.5.3 Loop Mode

When LOOPS is set, the RSRC bit in the same register chooses between loop mode (RSRC = 0) or single-wire mode (RSRC = 1). Loop mode is sometimes used to check software, independent of connections in the external system, to help isolate system problems. In this mode, the transmitter output is internally connected to the receiver input and the RxD pin is not used by the SCI, so it reverts to a general-purpose port I/O pin.

### 13.3.5.4 Single-Wire Operation

When LOOPS is set, the RSRC bit in the same register chooses between loop mode (RSRC = 0) or single-wire mode (RSRC = 1). Single-wire mode implements a half-duplex serial connection. The receiver is internally connected to the transmitter output and to the TxD pin. The RxD pin is not used and reverts to a general-purpose port I/O pin.

In single-wire mode, the TXDIR bit in SCIx C3 controls the direction of serial data on the TxD pin. When TXDIR is cleared, the TxD pin is an input to the SCI receiver and the transmitter is temporarily disconnected from the TxD pin so an external device can send serial data to the receiver. When TXDIR is set, the TxD pin is an output driven by the transmitter. In single-wire mode, the internal loop back connection from the transmitter to the receiver causes the receiver to receive characters that are sent out by the transmitter.

# Chapter 14

## Serial Peripheral Interface (SPI)

### 14.1 Introduction

The serial peripheral interface (SPI) module provides for full-duplex, synchronous, serial communication between the MCU and peripheral devices. These peripheral devices can include other microcontrollers, analog-to-digital converters, shift registers, sensors, memories, etc.

The SPI runs at a baud rate up to the bus clock divided by two in master mode and up to the bus clock divided by 4 in slave mode. Software can poll the status flags, or SPI operation can be interrupt driven.

#### NOTE

- MCF51CN128 series devices do not include stop1 low-power mode. Ignore references to stop1 in this chapter.
- Use pin mux control registers from [Section 2.3, “Pin Mux Controls”](#) to assign SPI signals to the MCF51CN128 package pins.
- Most pin functions default to GPIO and must be software configured before using SPI.

## 14.1.1 Features

Features of the SPI module include:

- Master or slave mode operation
- Full-duplex or single-wire bidirectional option
- Programmable transmit bit rate
- Double-buffered transmit and receive
- Serial clock phase and polarity options
- Slave select output
- Selectable MSB-first or LSB-first shifting

## 14.1.2 Block Diagrams

This section includes block diagrams showing SPI system connections, the internal organization of the SPI module, and the SPI clock dividers that control the master mode bit rate.

### 14.1.2.1 SPI System Block Diagram

[Figure 14-1](#) shows the SPI modules of two MCUs connected in a master-slave arrangement. The master device initiates all SPI data transfers. During a transfer, the master shifts data out (on the MOSI pin) to the slave while simultaneously shifting data in (on the MISO pin) from the slave. The transfer effectively exchanges the data that was in the SPI shift registers of the two SPI systems. The SPSCK signal is a clock output from the master and an input to the slave. The slave device must be selected by a low level on the slave select input ( $\overline{SS}$  pin). In this system, the master device has configured its  $\overline{SS}$  pin as an optional slave select output.

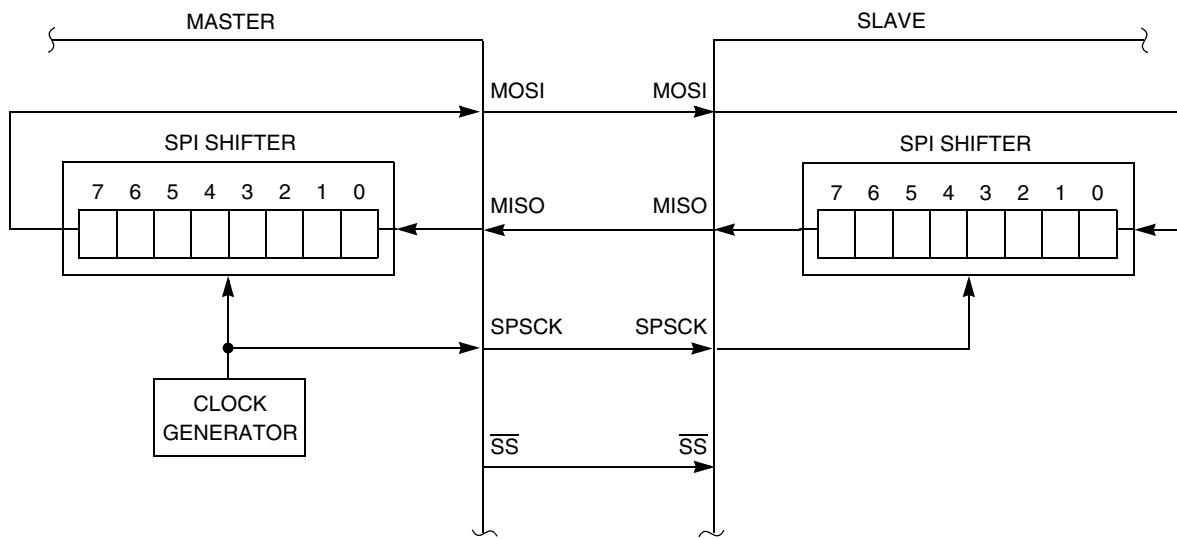


Figure 14-1. SPI System Connections

The most common uses of the SPI system include connecting simple shift registers for adding input or output ports or connecting small peripheral devices such as serial A/D or D/A converters. Although Figure 14-1 shows a system where data is exchanged between two MCUs, many practical systems involve simpler connections where data is unidirectionally transferred from the master MCU to a slave or from a slave to the master MCU.

#### 14.1.2.2 SPI Module Block Diagram

Figure 14-2 is a block diagram of the SPI module. The central element of the SPI is the SPI shift register. Data is written to the double-buffered transmitter (write to SPIxD) and gets transferred to the SPI shift register at the start of a data transfer. After shifting in a byte of data, the data is transferred into the double-buffered receiver where it can be read (read from SPIxD). Pin multiplexing logic controls connections between MCU pins and the SPI module.

When the SPI is configured as a master, the clock output is routed to the SPSCK pin, the shifter output is routed to MOSI, and the shifter input is routed from the MISO pin.

When the SPI is configured as a slave, the SPSCK pin is routed to the clock input of the SPI, the shifter output is routed to MISO, and the shifter input is routed from the MOSI pin.

In the external SPI system, simply connect all SPSCK pins to each other, all MISO pins together, and all MOSI pins together. Peripheral devices often use slightly different names for these pins.

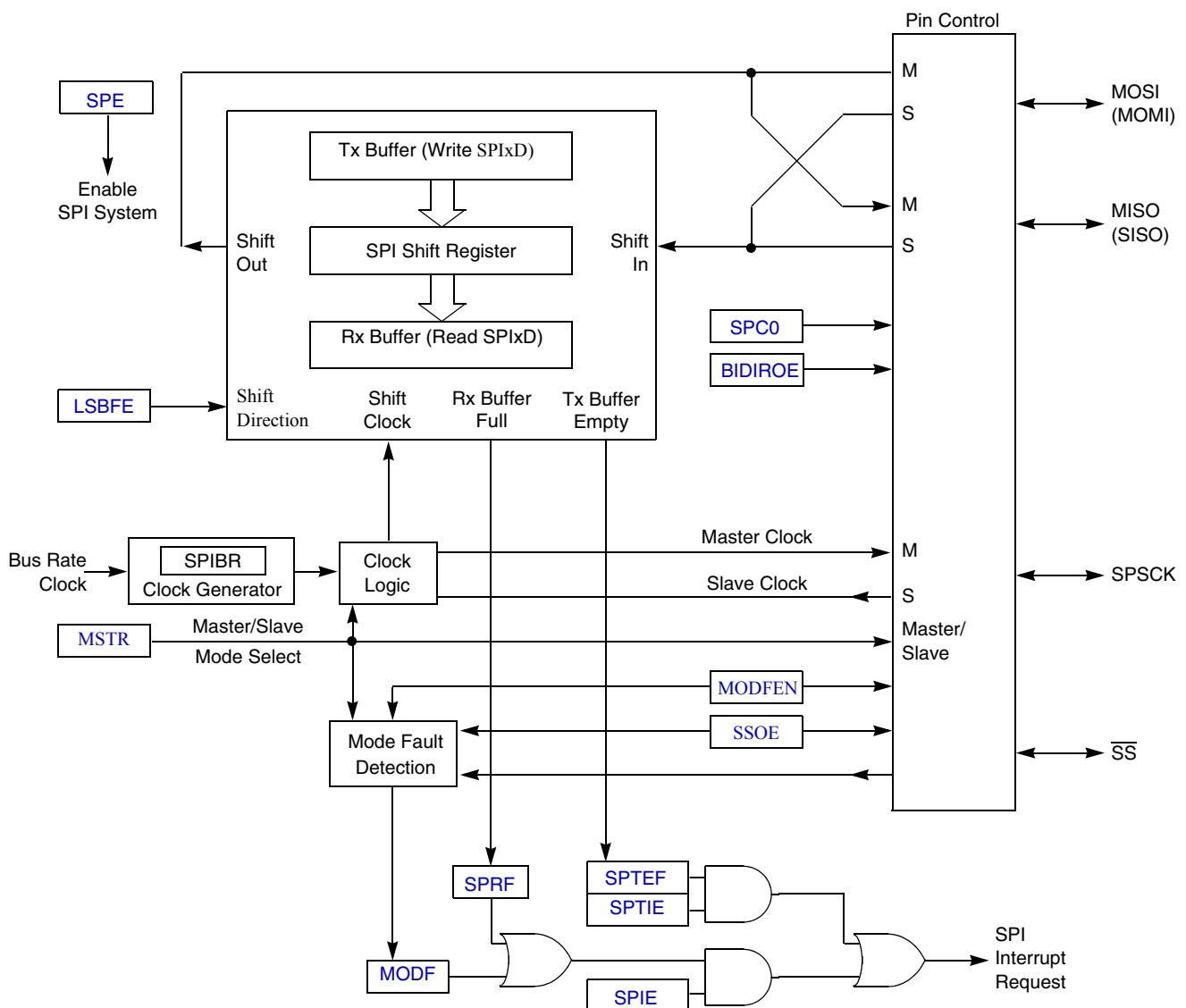


Figure 14-2. SPI Module Block Diagram

### 14.1.3 SPI Baud Rate Generation

As shown in Figure 14-3, the clock source for the SPI baud rate generator is the bus clock. The three prescale bits (SPPR2:SPPR1:SPPR0) choose a prescale divisor of 1, 2, 3, 4, 5, 6, 7, or 8. The three rate select bits (SPR2:SPR1:SPR0) divide the output of the prescaler stage by 2, 4, 8, 16, 32, 64, 128, or 256 to get the internal SPI master mode bit-rate clock.

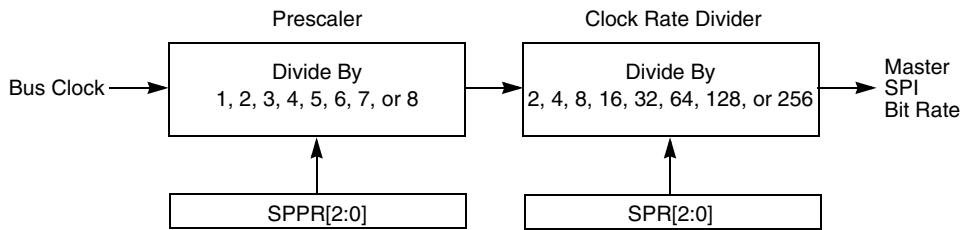


Figure 14-3. SPI Baud Rate Generation

## 14.2 External Signal Description

The SPI optionally shares four port pins. The function of these pins depends on the settings of SPI control bits. When the SPI is disabled ( $SPE = 0$ ), these four pins revert to being general-purpose port I/O pins that are not controlled by the SPI.

### 14.2.1 SPSCK — SPI Serial Clock

When the SPI is enabled as a slave, this pin is the serial clock input. When the SPI is enabled as a master, this pin is the serial clock output.

### 14.2.2 MOSI — Master Data Out, Slave Data In

When the SPI is enabled as a master and SPI pin control zero (SPC0) is 0 (not bidirectional mode), this pin is the serial data output. When the SPI is enabled as a slave and  $SPC0 = 0$ , this pin is the serial data input. If  $SPC0 = 1$  to select single-wire bidirectional mode, and master mode is selected, this pin becomes the bidirectional data I/O pin (MOMI). Also, the bidirectional mode output enable bit determines whether the pin acts as an input ( $BIDIROE = 0$ ) or an output ( $BIDIROE = 1$ ). If  $SPC0$  is set and slave mode is selected, this pin is not used by the SPI and reverts to being a general-purpose port I/O pin.

### 14.2.3 MISO — Master Data In, Slave Data Out

When the SPI is enabled as a master and SPI pin control zero (SPC0) is 0 (not bidirectional mode), this pin is the serial data input. When the SPI is enabled as a slave and  $SPC0$  is cleared, this pin is the serial data output. If  $SPC0$  is set to select single-wire bidirectional mode, and slave mode is selected, this pin becomes the bidirectional data I/O pin (SISO) and the bidirectional mode output enable bit determines whether the pin acts as an input ( $BIDIROE = 0$ ) or an output ( $BIDIROE = 1$ ). If  $SPC0$  is set and master mode is selected, this pin is not used by the SPI and reverts to being a general-purpose port I/O pin.

### 14.2.4 SS — Slave Select

When the SPI is enabled as a slave, this pin is the low-true slave select input. When the SPI is enabled as a master and mode fault enable is off ( $MODFEN = 0$ ), this pin is not used by the SPI and reverts to being a general-purpose port I/O pin. When the SPI is enabled as a master and  $MODFEN$  is set, the slave select output enable bit determines whether this pin acts as the mode fault input ( $SSOE = 0$ ) or as the slave select output ( $SSOE = 1$ ).

## 14.3 Modes of Operation

### 14.3.1 SPI in Stop Modes

The SPI is disabled in all stop modes, regardless of the settings before executing the STOP instruction. During either stop1 or stop2 mode, the SPI module is fully powered down. Upon wake-up from stop1 or stop2 mode, the SPI module is in the reset state. During stop3 mode, clocks to the SPI module are halted. No registers are affected. If stop3 is exited with a reset, the SPI is put into its reset state. If stop3 is exited with an interrupt, the SPI continues from the state it was in when stop3 was entered.

## 14.4 Register Definition

The SPI has five 8-bit registers to select SPI options, control baud rate, report SPI status, and for transmit/receive data.

Refer to the direct-page register summary in the [Chapter 4, “Memory”](#) for the absolute address assignments for all SPI registers. This section refers to registers and control bits only by their names, and a Freescale-provided equate or header file is used to translate these names into the appropriate absolute addresses.

### 14.4.1 SPI Control Register 1 (SPIxC1)

This read/write register includes the SPI enable control, interrupt enables, and configuration options.

	7	6	5	4		3	2	1	0
R W	SPIE	SPE	SPTIE	MSTR	CPOL	CPHA	SSOE	LSBFE	
Reset	0	0	0	0	0	1	0	0	0

Figure 14-4. SPI Control Register 1 (SPIxC1)

Table 14-1. SPIxC1 Field Descriptions

Field	Description
7 SPIE	<b>SPI Interrupt Enable (for SPRF and MODF)</b> — This is the interrupt enable for SPI receive buffer full (SPRF) and mode fault (MODF) events. 0 Interrupts from SPRF and MODF inhibited (use polling) 1 When SPRF or MODF is 1, request a hardware interrupt
6 SPE	<b>SPI System Enable</b> — Disabling the SPI halts any transfer that is in progress, clears data buffers, and initializes internal state machines. SPRF is cleared and SPTEF is set to indicate the SPI transmit data buffer is empty. 0 SPI system inactive 1 SPI system enabled
5 SPTIE	<b>SPI Transmit Interrupt Enable</b> — This is the interrupt enable bit for SPI transmit buffer empty (SPTEF). 0 Interrupts from SPTEF inhibited (use polling) 1 When SPTEF is 1, hardware interrupt requested

**Table 14-1. SPIxC1 Field Descriptions (continued)**

Field	Description
4 MSTR	<b>Master/Slave Mode Select</b> 0 SPI module configured as a slave SPI device 1 SPI module configured as a master SPI device
3 CPOL	<b>Clock Polarity</b> — This bit effectively places an inverter in series with the clock signal from a master SPI or to a slave SPI device. Refer to <a href="#">Section 14.5.1, “SPI Clock Formats”</a> for more details. 0 Active-high SPI clock (idle low) 1 Active-low SPI clock (idle high)
2 CPHA	<b>Clock Phase</b> — This bit selects one of two clock formats for different kinds of synchronous serial peripheral devices. Refer to <a href="#">Section 14.5.1, “SPI Clock Formats”</a> for more details. 0 First edge on SPSCK occurs at the middle of the first cycle of an 8-cycle data transfer 1 First edge on SPSCK occurs at the start of the first cycle of an 8-cycle data transfer
1 SSOE	<b>Slave Select Output Enable</b> — This bit is used in combination with the mode fault enable (MODFEN) bit in SPCR2 and the master/slave (MSTR) control bit to determine the function of the $\overline{SS}$ pin as shown in <a href="#">Table 14-2</a> .
0 LSBFE	<b>LSB First (Shifter Direction)</b> 0 SPI serial data transfers start with most significant bit 1 SPI serial data transfers start with least significant bit

**Table 14-2.  $\overline{SS}$  Pin Function**

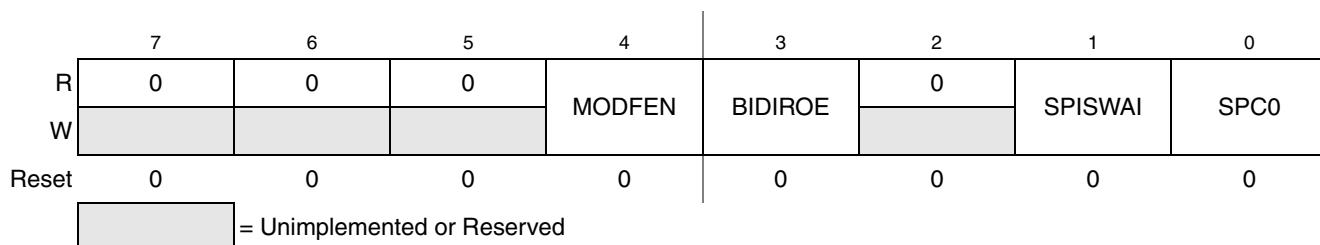
MODFEN	SSOE	Master Mode	Slave Mode
0	0	General-purpose I/O (not SPI)	Slave select input
0	1	General-purpose I/O (not SPI)	Slave select input
1	0	$\overline{SS}$ input for mode fault	Slave select input
1	1	Automatic $\overline{SS}$ output	Slave select input

**NOTE**

Ensure that the SPI must not be disabled (SPE=0) at the same time as a bit change to the CPHA bit. These changes must be performed as separate operations or unexpected behavior may occur.

#### 14.4.2 SPI Control Register 2 (SPIxC2)

This read/write register is used to control optional features of the SPI system. Bits 7, 6, 5, and 2 are not implemented and always read 0.

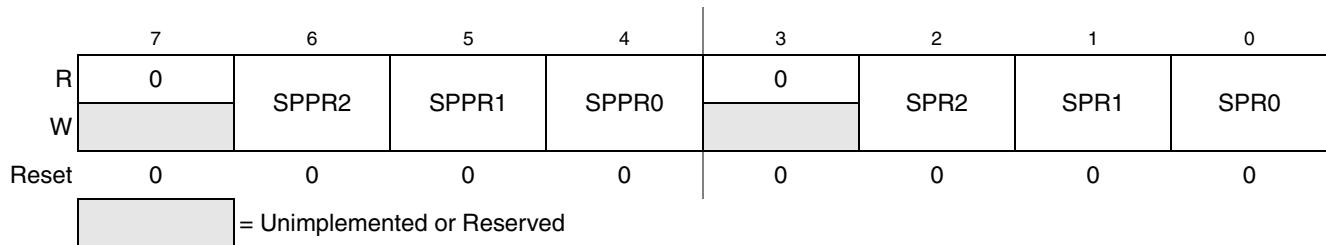
**Figure 14-5. SPI Control Register 2 (SPIxC2)**

**Table 14-3. SPIxC2 Register Field Descriptions**

Field	Description
4 MODFEN	<b>Master Mode-Fault Function Enable</b> — When the SPI is configured for slave mode, this bit has no meaning or effect. (The SS pin is the slave select input.) In master mode, this bit determines how the SS pin is used (refer to <a href="#">Table 14-2</a> for more details). 0 Mode fault function disabled, master SS pin reverts to general-purpose I/O not controlled by SPI 1 Mode fault function enabled, master SS pin acts as the mode fault input or the slave select output
3 BIDIROE	<b>Bidirectional Mode Output Enable</b> — When bidirectional mode is enabled by SPI pin control 0 (SPC0) = 1, BIDIROE determines whether the SPI data output driver is enabled to the single bidirectional SPI I/O pin. Depending on whether the SPI is configured as a master or a slave, it uses either the MOSI (MOMI) or MISO (SISO) pin, respectively, as the single SPI data I/O pin. When SPC0 = 0, BIDIROE has no meaning or effect. 0 Output driver disabled so SPI data I/O pin acts as an input 1 SPI I/O pin enabled as an output
1 SPISWAI	<b>SPI Stop in Wait Mode</b> 0 SPI clocks continue to operate in wait mode 1 SPI clocks stop when the MCU enters wait mode
0 SPC0	<b>SPI Pin Control 0</b> — The SPC0 bit chooses single-wire bidirectional mode. If MSTR = 0 (slave mode), the SPI uses the MISO (SISO) pin for bidirectional SPI data transfers. If MSTR is set (master mode), the SPI uses the MOSI (MOMI) pin for bidirectional SPI data transfers. When SPC0 is set, BIDIROE is used to enable or disable the output driver for the single bidirectional SPI I/O pin. 0 SPI uses separate pins for data input and data output 1 SPI configured for single-wire bidirectional operation

### 14.4.3 SPI Baud Rate Register (SPIxBR)

This register is used to set the prescaler and bit rate divisor for an SPI master. This register may be read or written at any time.

**Figure 14-6. SPI Baud Rate Register (SPIxBR)****Table 14-4. SPIxBR Register Field Descriptions**

Field	Description
6:4 SPPR[2:0]	<b>SPI Baud Rate Prescale Divisor</b> — This 3-bit field selects one of eight divisors for the SPI baud rate prescaler as shown in <a href="#">Table 14-5</a> . The input to this prescaler is the bus rate clock (BUSCLK). The output of this prescaler drives the input of the SPI baud rate divider (see <a href="#">Figure 14-3</a> ).
2:0 SPR[2:0]	<b>SPI Baud Rate Divider</b> — This field selects one of eight divisors for the SPI baud rate divider as shown in <a href="#">Table 14-6</a> . The input to this divider comes from the SPI baud rate prescaler (see <a href="#">Figure 14-3</a> ). The output of this divider is the SPI bit rate clock for master mode.

**Table 14-5. SPI Baud Rate Prescaler Divisor**

SPPR2:SPPR1:SPPR0	Prescaler Divisor
0:0:0	1
0:0:1	2
0:1:0	3
0:1:1	4
1:0:0	5
1:0:1	6
1:1:0	7
1:1:1	8

**Table 14-6. SPI Baud Rate Divisor**

SPR2:SPR1:SPR0	Rate Divisor
0:0:0	2
0:0:1	4
0:1:0	8
0:1:1	16
1:0:0	32
1:0:1	64
1:1:0	128
1:1:1	256

#### 14.4.4 SPI Status Register (SPIxS)

This register has three read-only status bits. Bits 6, 3, 2, 1, and 0 are not implemented and always read 0. Writes have no meaning or effect.

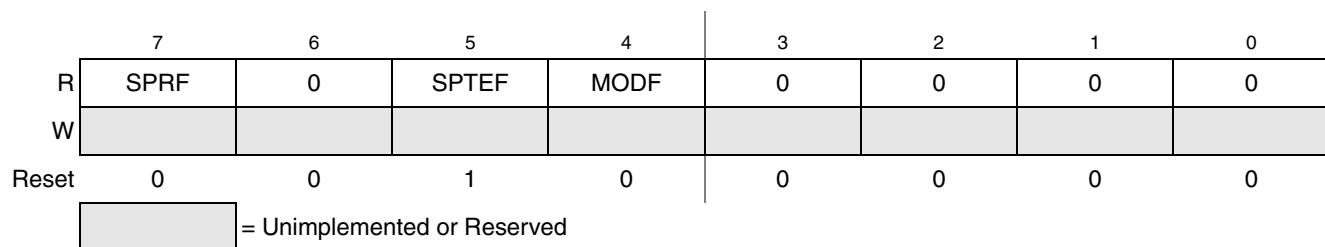
**Figure 14-7. SPI Status Register (SPIxS)**

Table 14-7. SPIxS Register Field Descriptions

Field	Description
7 SPRF	<b>SPI Read Buffer Full Flag</b> — SPRF is set at the completion of an SPI transfer to indicate that received data may be read from the SPI data register (SPIxD). SPRF is cleared by reading SPRF while it is set, then reading the SPI data register. 0 No data available in the receive data buffer 1 Data available in the receive data buffer
5 SPTEF	<b>SPI Transmit Buffer Empty Flag</b> — This bit is set when there is room in the transmit data buffer. It is cleared by reading SPIxS with SPTEF set, followed by writing a data value to the transmit buffer at SPIxD. SPIxS must be read with SPTEF set, before writing data to SPIxD or the SPIxD write is ignored. SPTEF generates an SPTEF CPU interrupt request if the SPTIE bit in the SPIxC1 is also set. SPTEF is automatically set when a data byte transfers from the transmit buffer into the transmit shift register. For an idle SPI (no data in the transmit buffer or the shift register and no transfer in progress), data written to SPIxD is transferred to the shifter almost immediately so SPTEF is set within two bus cycles allowing a second 8-bit data value to be queued into the transmit buffer. After completion of the transfer of the value in the shift register, the queued value from the transmit buffer automatically moves to the shifter and SPTEF is set to indicate there is room for new data in the transmit buffer. If no new data is waiting in the transmit buffer, SPTEF simply remains set and no data moves from the buffer to the shifter. 0 SPI transmit buffer not empty 1 SPI transmit buffer empty
4 MODF	<b>Master Mode Fault Flag</b> — MODF is set if the SPI is configured as a master and the slave select input goes low, indicating some other SPI device is also configured as a master. The SS pin acts as a mode fault error input only when MSTR and MODFEN are set and SSOE is cleared; otherwise, MODF is never set. MODF is cleared by reading MODF while it is 1, then writing to SPI control register 1 (SPIxC1). 0 No mode fault error 1 Mode fault error detected

#### 14.4.5 SPI Data Register (SPIxD)

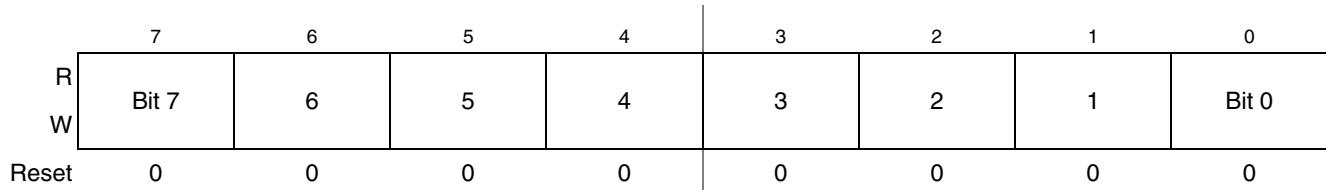


Figure 14-8. SPI Data Register (SPIxD)

Reads of this register return the data read from the receive data buffer. Writes to this register write data to the transmit data buffer. When the SPI is configured as a master, writing data to the transmit data buffer initiates an SPI transfer.

Data must not be written to the transmit data buffer unless the SPI transmit buffer empty flag (SPTEF) is set, indicating there is room in the transmit buffer to queue a new transmit byte.

Data may be read from SPIxD any time after SPRF is set and before another transfer is finished. Failure to read the data out of the receive data buffer before a new transfer ends causes a receive overrun condition and the data from the new transfer is lost.

## 14.5 Functional Description

An SPI transfer is initiated by checking for the SPI transmit buffer empty flag (SPTEF = 1) and then writing a byte of data to the SPI data register (SPIxD) in the master SPI device. When the SPI shift register is available, this byte of data is moved from the transmit data buffer to the shifter, SPTEF is set to indicate there is room in the buffer to queue another transmit character if desired, and the SPI serial transfer starts.

During the SPI transfer, data is sampled (read) on the MISO pin at one SPSCK edge and shifted, changing the bit value on the MOSI pin, one-half SPSCK cycle later. After eight SPSCK cycles, the data that was in the shift register of the master has been shifted out the MOSI pin to the slave while eight bits of data were shifted in the MISO pin into the master's shift register. At the end of this transfer, the received data byte is moved from the shifter into the receive data buffer and SPRF is set to indicate the data can be read by reading SPIxD. If another byte of data is waiting in the transmit buffer at the end of a transfer, it is moved into the shifter, SPTEF is set, and a new transfer is started.

Normally, SPI data is transferred most significant bit (MSB) first. If the least significant bit first enable (LSBFE) bit is set, SPI data is shifted LSB first.

When the SPI is configured as a slave, its  $\overline{SS}$  pin must be driven low before a transfer starts and  $\overline{SS}$  must stay low throughout the transfer. If a clock format where CPHA = 0 is selected,  $\overline{SS}$  must be driven to a logic 1 between successive transfers. If CPHA is set,  $\overline{SS}$  may remain low between successive transfers. See [Section 14.5.1, “SPI Clock Formats”](#) for more details.

Because the transmitter and receiver are double buffered, a second byte, in addition to the byte currently being shifted out, can be queued into the transmit data buffer, and a previously received character can be in the receive data buffer while a new character is being shifted in. The SPTEF flag indicates when the transmit buffer has room for a new character. The SPRF flag indicates when a received character is available in the receive data buffer. The received character must be read out of the receive buffer (read SPIxD) before the next transfer is finished or a receive overrun error results.

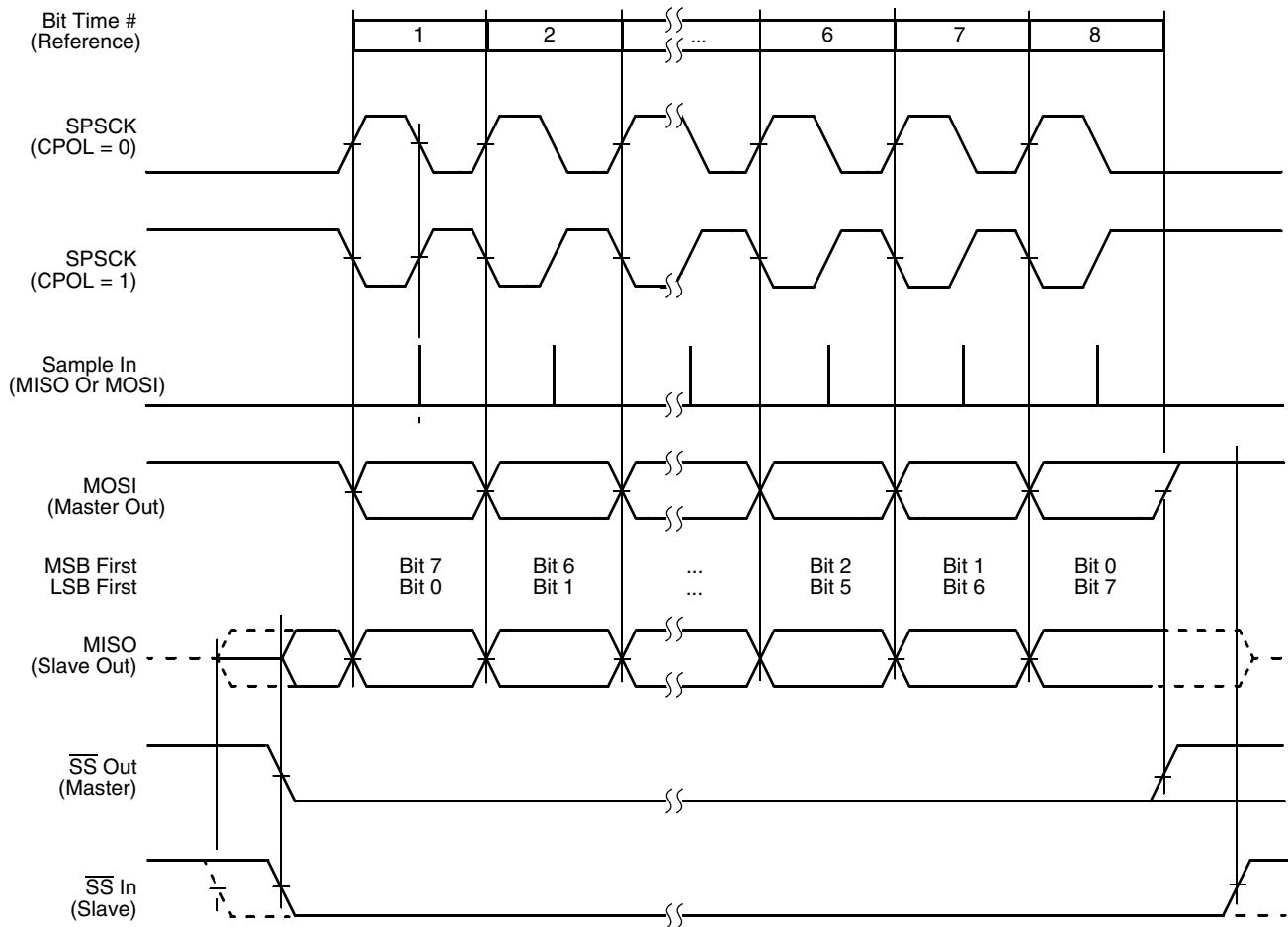
In the case of a receive overrun, the new data is lost because the receive buffer still held the previous character and was not ready to accept the new data. There is no indication for such an overrun condition so the application system designer must ensure that previous data has been read from the receive buffer before a new transfer is initiated.

### 14.5.1 SPI Clock Formats

To accommodate a wide variety of synchronous serial peripherals from different manufacturers, the SPI system has a clock polarity (CPOL) bit and a clock phase (CPHA) control bit to select one of four clock formats for data transfers. CPOL selectively inserts an inverter in series with the clock. CPHA chooses between two different clock phase relationships between the clock and data.

[Figure 14-9](#) shows the clock formats when CPHA is set. At the top of the figure, the eight bit times are shown for reference with bit 1 starting at the first SPSCK edge and bit 8 ending one-half SPSCK cycle after the sixteenth SPSCK edge. The MSB first and LSB first lines show the order of SPI data bits depending on the setting in LSBFE. Both variations of SPSCK polarity are shown, but only one of these waveforms applies for a specific transfer, depending on the value in CPOL. The SAMPLE IN waveform applies to the MOSI input of a slave or the MISO input of a master. The MOSI waveform applies to the

MOSI output pin from a master and the MISO waveform applies to the MISO output from a slave. The  $\overline{SS}$  OUT waveform applies to the slave select output from a master (provided MODFEN and SSOE = 1). The master  $\overline{SS}$  output goes to active low one-half SPSCK cycle before the start of the transfer and goes back high at the end of the eighth bit time of the transfer. The  $\overline{SS}$  IN waveform applies to the slave select input of a slave.



**Figure 14-9. SPI Clock Formats (CPHA = 1)**

When CPHA is set, the slave begins to drive its MISO output when  $\overline{SS}$  goes to active low, but the data is not defined until the first SPSCK edge. The first SPSCK edge shifts the first bit of data from the shifter onto the MOSI output of the master and the MISO output of the slave. The next SPSCK edge causes both the master and the slave to sample the data bit values on their MISO and MOSI inputs, respectively. At the third SPSCK edge, the SPI shifter shifts one bit position which shifts in the bit value that was just sampled, and shifts the second data bit value out the other end of the shifter to the MOSI and MISO outputs of the master and slave, respectively. When CHPA is set, the slave's  $\overline{SS}$  input is not required to go to its inactive high level between transfers.

Figure 14-10 shows the clock formats when CPHA is cleared. At the top of the figure, the eight bit times are shown for reference with bit 1 starting as the slave is selected ( $\overline{SS}$  IN goes low), and bit 8 ends at the last SPSCK edge. The MSB first and LSB first lines show the order of SPI data bits depending on the

setting in LSBFE. Both variations of SPSCK polarity are shown, but only one of these waveforms applies for a specific transfer, depending on the value in CPOL. The Sample In waveform applies to the MOSI input of a slave or the MISO input of a master. The MOSI waveform applies to the MOSI output pin from a master and the MISO waveform applies to the MISO output from a slave. The SS OUT waveform applies to the slave select output from a master (provided MODFEN and SSOE = 1). The master SS output goes to active low at the start of the first bit time of the transfer and goes back high one-half SPSCK cycle after the end of the eighth bit time of the transfer. The SS IN waveform applies to the slave select input of a slave.

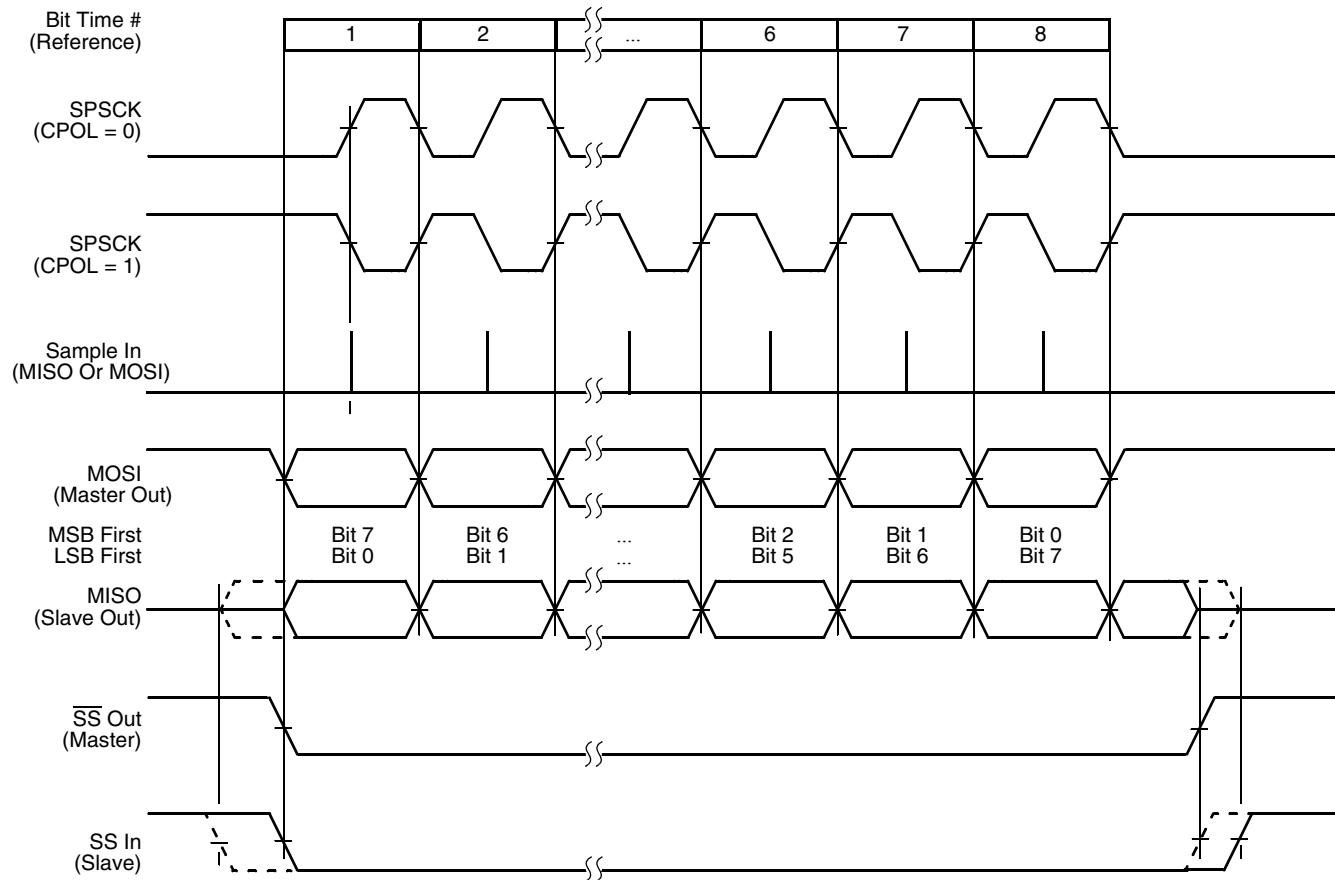


Figure 14-10. SPI Clock Formats (CPHA = 0)

When CPHA = 0, the slave begins to drive its MISO output with the first data bit value (MSB or LSB depending on LSBFE) when SS goes to active low. The first SPSCK edge causes both the master and the slave to sample the data bit values on their MISO and MOSI inputs, respectively. At the second SPSCK edge, the SPI shifter shifts one bit position which shifts in the bit value that was just sampled and shifts the second data bit value out the other end of the shifter to the MOSI and MISO outputs of the master and slave, respectively. When CPHA is cleared, the slave's SS input must go to its inactive high level between transfers.

### 14.5.2 SPI Interrupts

There are three flag bits, two interrupt mask bits, and one interrupt vector associated with the SPI system. The SPI interrupt enable mask (SPIE) enables interrupts from the SPI receiver full flag (SPRF) and mode

fault flag (MODF). The SPI transmit interrupt enable mask (SPTIE) enables interrupts from the SPI transmit buffer empty flag (SPTEF). When one of the flag bits is set, and the associated interrupt mask bit is set, a hardware interrupt request is sent to the CPU. If the interrupt mask bits are cleared, software can poll the associated flag bits instead of using interrupts. The SPI interrupt service routine (ISR) must check the flag bits to determine what event caused the interrupt. The service routine must also clear the flag bit(s) before returning from the ISR (usually near the beginning of the ISR).

### 14.5.3 Mode Fault Detection

A mode fault occurs and the mode fault flag (MODF) becomes set when a master SPI device detects an error on the  $\overline{SS}$  pin (provided the  $\overline{SS}$  pin is configured as the mode fault input signal). The  $\overline{SS}$  pin is configured to be the mode fault input signal when MSTR is set, mode fault enable is set (MODFEN = 1), and slave select output enable is cleared (SSOE = 0).

The mode fault detection feature can be used in a system where more than one SPI device might become a master at the same time. The error is detected when a master's  $\overline{SS}$  pin is low, indicating that some other SPI device is trying to address this master as if it were a slave. This could indicate a harmful output driver conflict, so the mode fault logic is designed to disable all SPI output drivers when such an error is detected.

When a mode fault is detected, MODF is set and MSTR is cleared to change the SPI configuration back to slave mode. The output drivers on the SPSCK, MOSI, and MISO (if not bidirectional mode) are disabled.

MODF is cleared by reading it while it is set, then writing to the SPI control register 1 (SPIxC1). User software must verify the error condition has been corrected before changing the SPI back to master mode.

# Chapter 15

## Analog-to-Digital Converter (ADC12)

### 15.1 Introduction

The 12-bit analog-to-digital converter (ADC) is a successive approximation ADC designed for operation within an integrated microcontroller system-on-chip.

#### NOTE

- MCF51CN128 series devices do not include stop1 mode and pin control registers. Ignore references to stop1 and pin control registers. For details on low-power mode operation, refer to [Table 3-5](#) in [Chapter 3, “Modes of Operation”](#). For details on register memory map, refer to [Table 4-3](#) in [Chapter 4, “Memory”](#).
- Use pin mux control registers from [Section 2.3, “Pin Mux Controls”](#) to assign ADC signals to the MCF51CN128 package pins.
- Most pin functions default to GPIO and must be software configured before using ADC.

#### 15.1.1 ADC Clock Gating

The bus clock to the ADC can be gated on and off using the SCGC1[ADC] bit (see [Section 5.7.10, “System Clock Gating Control 1 Register \(SCGC1\)”](#)). This bit is set after any reset that enables the bus clock to this module. To conserve power, the SCGC1[ADC] bit can be cleared to disable the clock to this module when not in use. See [Section 5.6, “Peripheral Clock Gating,”](#) for details.

#### 15.1.2 Module Configurations

This section provides information for configuring the ADC on this device.

##### 15.1.2.1 Channel Assignments

The ADC channel assignments for this device are shown in [Table 15-1](#). Reserved channels convert to an unknown value.

**Table 15-1. ADC Channel Assignment**

<b>ADCH</b>	<b>Channel</b>	<b>Input</b>	<b>ADCH</b>	<b>Channel</b>	<b>Input</b>
00000	AD0	PTE2/KBI2P2/SS2/ADP0	10000	AD16	Reserved
00001	AD1	PTE1/KBI2P1/MOSI2/ADP1	10001	AD17	Reserved
00010	AD2	PTE0/KBI2P0/MISO2/ADP2	10010	AD18	Reserved
00011	AD3	PTD7GPIO7/SPSCK2/ADP3	10011	AD19	Reserved
00100	AD4	PTD3GPIO3/RXD2/ADP4	10100	AD20	Reserved
00101	AD5	PTD2/GPIO2/TXD2/ADP5	10101	AD21	Reserved
00110	AD6	PTD1/GPIO1/RXD1/ADP6	10110	AD22	Reserved
00111	AD7	PTD0/GPIO0/TXD1/ADP7	10111	AD23	switched digital core supply
01000	AD8	PTC7/SDA2/SPSCK1/ADP8	11000	AD24	unswitched analog supply
01001	AD9	PTC6/SCL2/MISO1/ADP9	11001	AD25	unswitched digital core supply
01010	AD10	PTC5/MOSI1/ADP10	11010	AD26	Temperature Sensor <sup>1</sup>
01011	AD11	PTC4/IRQ/SS1/ADP11	11011	AD27	Internal Bandgap
01100	AD12	Reserved	11100	—	Reserved
01101	AD13	Reserved	11101	V <sub>REFH</sub>	V <sub>DD</sub>
01110	AD14	Reserved	11110	V <sub>REFL</sub>	V <sub>SS</sub>
01111	AD15	Reserved	11111	Module Disabled	None

<sup>1</sup> For information, see [Section 15.1.2.4, “Temperature Sensor.”](#)

### NOTE

Selecting the internal bandgap channel requires SPMSC1[BGBE] to be set (see [Section 5.7.7, “System Power Management Status and Control 1 Register \(SPMSC1\)”](#)). See *MCF51CN128 Data Sheet* for the value of the bandgap voltage.

#### 15.1.2.2 Alternate Clock

The ADC is capable of performing conversions using the MCU bus clock, the bus clock divided by two, the local asynchronous clock (ADACK) within the module, or the alternate clock (ALTCLK). The ALTCLK on this device is the MGERCLK. See [Chapter 6, “Multipurpose Clock Generator \(MCG\),”](#) for more information.

### 15.1.2.3 Hardware Trigger

The RTC on this device can be enabled as a hardware trigger for the ADC module by setting the ADCSC2[ADTRG] bit. When enabled, the ADC is triggered every time RTCCNT matches RTCMOD. The RTC interrupt does not have to be enabled to trigger the ADC.

The RTI can be configured to cause a hardware trigger in MCU run, wait, and stop3.

### 15.1.2.4 Temperature Sensor

The ADC module includes a temperature sensor whose output is connected to one of the ADC analog channel inputs. [Equation 15-1](#) provides an approximate transfer function of the temperature sensor.

$$\text{Temp} = 25 - ((V_{\text{TEMP}} - V_{\text{TEMP25}}) \div m) \quad \text{Eqn. 15-1}$$

where:

- $V_{\text{TEMP}}$  is the voltage of the temperature sensor channel at the ambient temperature.
- $V_{\text{TEMP25}}$  is the voltage of the temperature sensor channel at 25°C.
- $m$  is the hot or cold voltage versus temperature slope in V/°C.

For temperature calculations, use the  $V_{\text{TEMP25}}$  and  $m$  values in the data sheet.

### 15.1.3 Features

Features of the ADC module include:

- Linear successive approximation algorithm with 12-bit resolution
- Up to 28 analog inputs
- Output formatted in 12-, 10-, or 8-bit right-justified unsigned format
- Single or continuous conversion (automatic return to idle after single conversion)
- Configurable sample time and conversion speed/power
- Conversion complete flag and interrupt
- Input clock selectable from up to four sources
- Operation in wait or stop3 modes for lower noise operation
- Asynchronous clock source for lower noise operation
- Selectable asynchronous hardware conversion trigger
- Automatic compare with interrupt for less-than, or greater-than or equal-to, programmable value

### 15.1.4 Block Diagram

Figure 15-1 provides a block diagram of the ADC module.

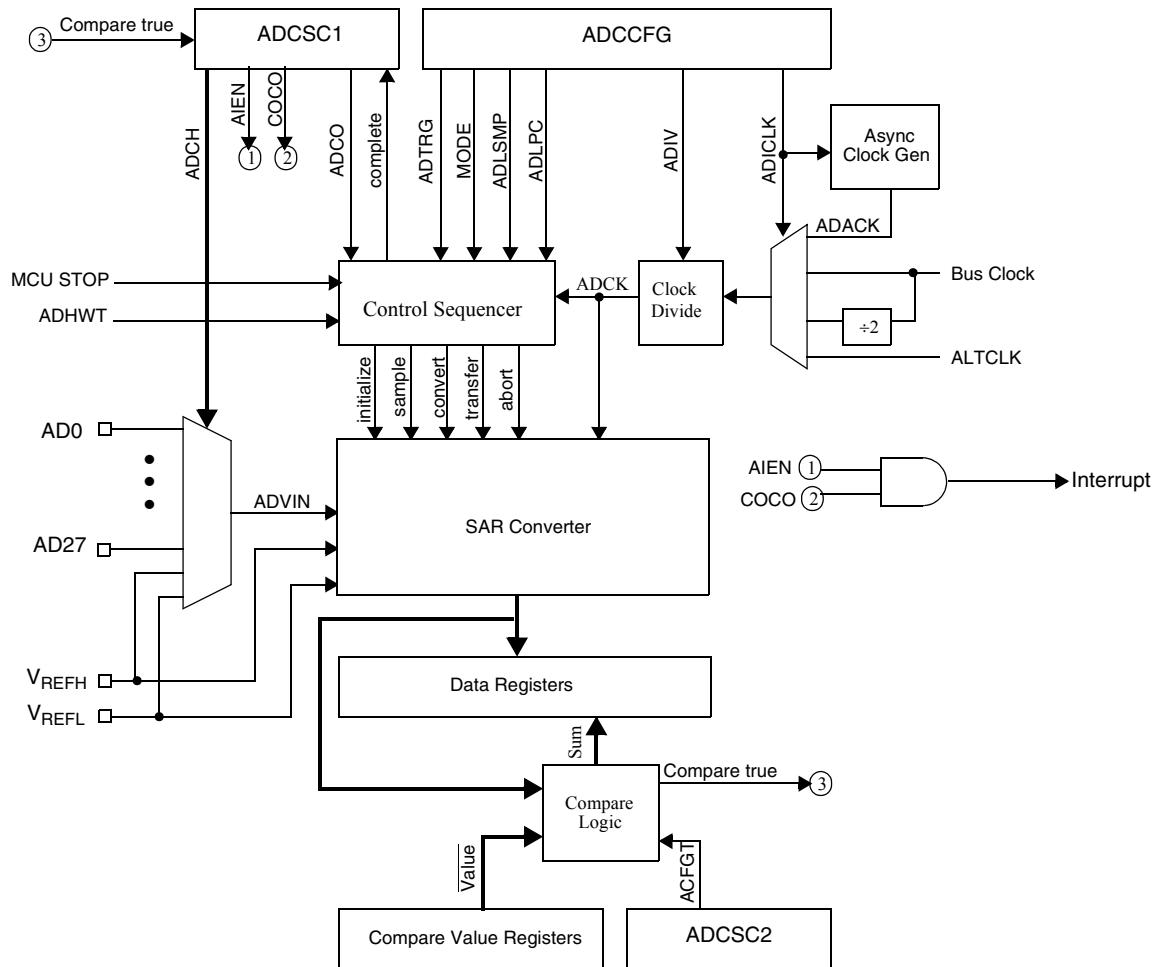


Figure 15-1. ADC Block Diagram

## 15.2 External Signal Description

The ADC module supports up to 28 separate analog inputs. It also requires four supply/reference/ground connections.

Table 15-2. Signal Properties

Name	Function
AD27–AD0	Analog Channel inputs
V <sub>REFH</sub>	High reference voltage
V <sub>REFL</sub>	Low reference voltage
V <sub>DDAD</sub>	Analog power supply
V <sub>SSAD</sub>	Analog ground

### 15.2.1 Analog Power ( $V_{DDAD}$ )

The ADC analog portion uses  $V_{DDAD}$  as its power connection. In some packages,  $V_{DDAD}$  is connected internally to  $V_{DD}$ . If externally available, connect the  $V_{DDAD}$  pin to the same voltage potential as  $V_{DD}$ . External filtering may be necessary to ensure clean  $V_{DDAD}$  for good results.

### 15.2.2 Analog Ground ( $V_{SSAD}$ )

The ADC analog portion uses  $V_{SSAD}$  as its ground connection. In some packages,  $V_{SSAD}$  is connected internally to  $V_{SS}$ . If externally available, connect the  $V_{SSAD}$  pin to the same voltage potential as  $V_{SS}$ .

### 15.2.3 Voltage Reference High ( $V_{REFH}$ )

$V_{REFH}$  is the high reference voltage for the converter. In some packages,  $V_{REFH}$  is connected internally to  $V_{DDAD}$ . If externally available,  $V_{REFH}$  may be connected to the same potential as  $V_{DDAD}$  or may be driven by an external source between the minimum  $V_{DDAD}$  spec and the  $V_{DDAD}$  potential ( $V_{REFH}$  must never exceed  $V_{DDAD}$ ).

### 15.2.4 Voltage Reference Low ( $V_{REFL}$ )

$V_{REFL}$  is the low-reference voltage for the converter. In some packages,  $V_{REFL}$  is connected internally to  $V_{SSAD}$ . If externally available, connect the  $V_{REFL}$  pin to the same voltage potential as  $V_{SSAD}$ .

### 15.2.5 Analog Channel Inputs (ADx)

The ADC module supports up to 28 separate analog inputs. An input is selected for conversion through the ADCH channel select bits.

## 15.3 Register Definition

These memory-mapped registers control and monitor operation of the ADC:

- Status and control register, ADCSC1
- Status and control register, ADCSC2
- Data result registers, ADCRH and ADCRL
- Compare value registers, ADCCVH and ADCCVL
- Configuration register, ADCCFG

### 15.3.1 Status and Control Register 1 (ADCSC1)

This section describes the function of the ADC status and control register (ADCSC1). Writing ADCSC1 aborts the current conversion and initiates a new conversion (if the ADCH bits are equal to a value other than all 1s).



Figure 15-2. Status and Control Register (ADCSC1)

Table 15-3. ADCSC1 Field Descriptions

Field	Description
7 COCO	Conversion Complete Flag. The COCO flag is a read-only bit set each time a conversion is completed when the compare function is disabled (ACFE = 0). When the compare function is enabled (ACFE = 1), the COCO flag is set upon completion of a conversion only if the compare result is true. This bit is cleared when ADCSC1 is written or when ADCRL is read. 0 Conversion not completed 1 Conversion completed
6 AIEN	Interrupt Enable AIEN enables conversion complete interrupts. When COCO becomes set while AIEN is high, an interrupt is asserted. 0 Conversion complete interrupt disabled 1 Conversion complete interrupt enabled
5 ADCO	Continuous Conversion Enable. ADCO enables continuous conversions. 0 One conversion following a write to the ADCSC1 when software triggered operation is selected, or one conversion following assertion of ADHWT when hardware triggered operation is selected. 1 Continuous conversions initiated following a write to ADCSC1 when software triggered operation is selected. Continuous conversions are initiated by an ADHWT event when hardware triggered operation is selected.
4:0 ADCH	Input Channel Select. The ADCH bits form a 5-bit field that selects one of the input channels. The input channels are detailed in <a href="#">Table 15-4</a> . The successive approximation converter subsystem is turned off when the channel select bits are all set. This feature allows for explicit disabling of the ADC and isolation of the input channel from all sources. Terminating continuous conversions this way prevents an additional, single conversion from being performed. It is not necessary to set the channel select bits to all ones to place the ADC in a low-power state when continuous conversions are not enabled because the module automatically enters a low-power state when a conversion completes.

Table 15-4. Input Channel Select

ADCH	Input Select
00000–01111	AD0–15
10000–11011	AD16–27
11100	Reserved
11101	V <sub>REFH</sub>
11110	V <sub>REFL</sub>
11111	Module disabled

### 15.3.2 Status and Control Register 2 (ADCSC2)

The ADCSC2 register controls the compare function, conversion trigger, and conversion active of the ADC module.

	7	6	5	4	3	2	1	0
R	ADACT	ADTRG	ACFE	ACFGT	0	0	R <sup>1</sup>	R <sup>1</sup>
W								
Reset:	0	0	0	0	0	0	0	0

Figure 15-3. Status and Control Register 2 (ADCSC2)

<sup>1</sup> Bits 1 and 0 are reserved bits that must always be written to 0.

Table 15-5. ADCSC2 Register Field Descriptions

Field	Description
7 ADACT	Conversion Active. Indicates that a conversion is in progress. ADACT is set when a conversion is initiated and cleared when a conversion is completed or aborted. 0 Conversion not in progress 1 Conversion in progress
6 ADTRG	Conversion Trigger Select. Selects the type of trigger used for initiating a conversion. Two types of trigger are selectable: software trigger and hardware trigger. When software trigger is selected, a conversion is initiated following a write to ADCSC1. When hardware trigger is selected, a conversion is initiated following the assertion of the ADHWT input. 0 Software trigger selected 1 Hardware trigger selected
5 ACFE	Compare Function Enable. Enables the compare function. 0 Compare function disabled 1 Compare function enabled
4 ACFGT	Compare Function Greater Than Enable. Configures the compare function to trigger when the result of the conversion of the input being monitored is greater than or equal to the compare value. The compare function defaults to triggering when the result of the compare of the input being monitored is less than the compare value. 0 Compare triggers when input is less than compare level 1 Compare triggers when input is greater than or equal to compare level

### 15.3.3 Data Result High Register (ADCRH)

In 12-bit operation, ADCRH contains the upper four bits of the result of a 12-bit conversion. In 10-bit mode, ADCRH contains the upper two bits of the result of a 10-bit conversion. When configured for 10-bit mode, ADR[11:10] are cleared. When configured for 8-bit mode, ADR11 – ADR8 are equal to zero.

In 12-bit and 10-bit mode, ADCRH is updated each time a conversion completes except when automatic compare is enabled and the compare condition is not met. In 12-bit and 10-bit mode, reading ADCRH prevents the ADC from transferring subsequent conversion results into the result registers until ADCRL is read. If ADCRL is not read until after the next conversion is completed, the intermediate conversion result is lost. In 8-bit mode, there is no interlocking with ADCRL.

If the MODE bits are changed, any data in ADCRH becomes invalid.

	7	6	5	4		3	2	1	0
R	0	0	0	0	ADR11	ADR10	ADR9	ADR8	
W									
Reset:	0	0	0	0	0	0	0	0	0

Figure 15-4. Data Result High Register (ADCRH)

### 15.3.4 Data Result Low Register (ADCRL)

ADCRL contains the lower eight bits of the result of a 12-bit or 10-bit conversion, and all eight bits of an 8-bit conversion. This register is updated each time a conversion completes except when automatic compare is enabled and the compare condition is not met. In 12-bit and 10-bit mode, reading ADCRH prevents the ADC from transferring subsequent conversion results into the result registers until ADCRL is read. If ADCRL is not read until the after next conversion is completed, the intermediate conversion results are lost. In 8-bit mode, there is no interlocking with ADCRH. If the MODE bits are changed, any data in ADCRL becomes invalid.

	7	6	5	4		3	2	1	0
R	ADR7	ADR6	ADR5	ADR4	ADR3	ADR2	ADR1	ADR0	
W									
Reset:	0	0	0	0	0	0	0	0	0

Figure 15-5. Data Result Low Register (ADCRL)

### 15.3.5 Compare Value High Register (ADCCVH)

In 12-bit mode, the ADCCVH register holds the upper four bits of the 12-bit compare value. These bits are compared to the upper four bits of the result following a conversion in 12-bit mode when the compare function is enabled.

	7	6	5	4		3	2	1	0
R	0	0	0	0	ADCV11	ADCV10	ADCV9	ADCV8	
W									
Reset:	0	0	0	0	0	0	0	0	0

Figure 15-6. Compare Value High Register (ADCCVH)

## Analog-to-Digital Converter (ADC12)

In 10-bit mode, the ADCCVH register holds the upper two bits of the 10-bit compare value (ADCV9 – ADCV8). These bits are compared to the upper two bits of the result following a conversion in 10-bit mode when the compare function is enabled.

In 8-bit mode, ADCCVH is not used during compare.

### 15.3.6 Compare Value Low Register (ADCCVL)

This register holds the lower 8 bits of the 12-bit or 10-bit compare value or all 8 bits of the 8-bit compare value. Bits ADCV7:ADCV0 are compared to the lower 8 bits of the result following a conversion in 12-bit, 10-bit or 8-bit mode.

	7	6	5	4		3	2	1	0
R W	ADCV7	ADCV6	ADCV5	ADCV4	ADCV3	ADCV2	ADCV1	ADCV0	
Reset:	0	0	0	0	0	0	0	0	0

Figure 15-7. Compare Value Low Register(ADCCVL)

### 15.3.7 Configuration Register (ADCCFG)

ADCCFG selects the mode of operation, clock source, clock divide, and configure for low power or long sample time.

	7	6	5	4		3	2	1	0
R W	ADLPC	ADIV	ADLSMP	MODE	ADICLK				
Reset:	0	0	0	0	0	0	0	0	0

Figure 15-8. Configuration Register (ADCCFG)

Table 15-6. ADCCFG Register Field Descriptions

Field	Description
7 ADLPC	Low-Power Configuration. ADLPC controls the speed and power configuration of the successive approximation converter. This optimizes power consumption when higher sample rates are not required. 0 High speed configuration 1 Low power configuration: The power is reduced at the expense of maximum clock speed.
6:5 ADIV	Clock Divide Select. ADIV selects the divide ratio used by the ADC to generate the internal clock ADCK. <a href="#">Table 15-7</a> shows the available clock configurations.
4 ADLSMP	Long Sample Time Configuration. ADLSMP selects between long and short sample time. This adjusts the sample period to allow higher impedance inputs to be accurately sampled or to maximize conversion speed for lower impedance inputs. Longer sample times can also be used to lower overall power consumption when continuous conversions are enabled if high conversion rates are not required. 0 Short sample time 1 Long sample time

**Table 15-6. ADCCFG Register Field Descriptions (continued)**

<b>Field</b>	<b>Description</b>
3:2 MODE	Conversion Mode Selection. MODE bits are used to select between 12-, 10-, or 8-bit operation. See <a href="#">Table 15-8</a> .
1:0 ADICLK	Input Clock Select. ADICLK bits select the input clock source to generate the internal clock ADCK. See <a href="#">Table 15-9</a> .

**Table 15-7. Clock Divide Select**

<b>ADIV</b>	<b>Divide Ratio</b>	<b>Clock Rate</b>
00	1	Input clock
01	2	Input clock $\div 2$
10	4	Input clock $\div 4$
11	8	Input clock $\div 8$

**Table 15-8. Conversion Modes**

<b>MODE</b>	<b>Mode Description</b>
00	8-bit conversion (N=8)
01	12-bit conversion (N=12)
10	10-bit conversion (N=10)
11	Reserved

**Table 15-9. Input Clock Select**

<b>ADICLK</b>	<b>Selected Clock Source</b>
00	Bus clock
01	Bus clock divided by 2
10	Alternate clock (ALTCLK)
11	Asynchronous clock (ADACK)

## 15.4 Functional Description

The ADC module is disabled during reset or when the ADCH bits are all high. The module is idle when a conversion has completed and another conversion has not been initiated. When idle, the module is in its lowest power state.

The ADC can perform an analog-to-digital conversion on any of the software selectable channels. In 12-bit and 10-bit mode, the selected channel voltage is converted by a successive approximation algorithm into a 12-bit digital result. In 8-bit mode, the selected channel voltage is converted by a successive approximation algorithm into a 9-bit digital result.

When the conversion is completed, the result is placed in the data registers (ADCRH and ADCRL). In 10-bit mode, the result is rounded to 10 bits and placed in the data registers (ADCRH and ADCRL). In 8-bit mode, the result is rounded to 8 bits and placed in ADCRL. The conversion complete flag (COCO) is then set and an interrupt is generated if the conversion complete interrupt has been enabled (AIEN = 1).

The ADC module has the capability of automatically comparing the result of a conversion with the contents of its compare registers. The compare function is enabled by setting the ACFE bit and operates with any of the conversion modes and configurations.

### 15.4.1 Clock Select and Divide Control

One of four clock sources can be selected as the clock source for the ADC module. This clock source is then divided by a configurable value to generate the input clock to the converter (ADCK). The clock is selected from one of the following sources by means of the ADICLK bits.

- The bus clock. This is the default selection following reset.
- The bus clock divided by two. For higher bus clock rates, this allows a maximum divide by 16 of the bus clock.
- ALTCLK, as defined for this MCU (See module section introduction).
- The asynchronous clock (ADACK). This clock is generated from a clock source within the ADC module. When selected as the clock source, this clock remains active while the MCU is in wait or stop3 mode and allows conversions in these modes for lower noise operation.

Whichever clock is selected, its frequency must fall within the specified frequency range for ADCK. If the available clocks are too slow, the ADC do not perform according to specifications. If the available clocks are too fast, the clock must be divided to the appropriate frequency. This divider is specified by the ADIV bits and can be divide-by 1, 2, 4, or 8.

### 15.4.2 Input Select and Pin Control

The pin control registers disable the I/O port control of the pins used as analog inputs. When a pin control register bit is set, the following conditions are forced for the associated MCU pin:

- The output buffer is forced to its high impedance state.
- The input buffer is disabled. A read of the I/O port returns a zero for any pin with its input buffer disabled.
- The pullup is disabled.

### 15.4.3 Hardware Trigger

The ADC module has a selectable asynchronous hardware conversion trigger, ADHWT, enabled when the ADTRG bit is set. This source is not available on all MCUs. Consult the module introduction for information on the ADHWT source specific to this MCU.

When ADHWT source is available and hardware trigger is enabled (ADTRG=1), a conversion is initiated on the rising edge of ADHWT. If a conversion is in progress when a rising edge occurs, the rising edge is ignored. In continuous convert configuration, only the initial rising edge to launch continuous conversions

is observed. The hardware trigger function operates in conjunction with any of the conversion modes and configurations.

## 15.4.4 Conversion Control

Conversions can be performed in 12-bit mode, 10-bit mode, or 8-bit mode as determined by the MODE bits. Conversions can be initiated by a software or hardware trigger. In addition, the ADC module can be configured for low power operation, long sample time, continuous conversion, and automatic compare of the conversion result to a software determined compare value.

### 15.4.4.1 Initiating Conversions

A conversion is initiated:

- Following a write to ADCSC1 (with ADCH bits not all 1s) if software triggered operation is selected.
- Following a hardware trigger (ADHWT) event if hardware triggered operation is selected.
- Following the transfer of the result to the data registers when continuous conversion is enabled.

If continuous conversions are enabled, a new conversion is automatically initiated after the completion of the current conversion. In software triggered operation, continuous conversions begin after ADCSC1 is written and continue until aborted. In hardware triggered operation, continuous conversions begin after a hardware trigger event and continue until aborted.

### 15.4.4.2 Completing Conversions

A conversion is completed when the result of the conversion is transferred into the data result registers, ADCRH and ADCRL. This is indicated by the setting of COCO. An interrupt is generated if AIEN is high at the time that COCO is set.

A blocking mechanism prevents a new result from overwriting previous data in ADCRH and ADCRL if the previous data is in the process of being read while in 12-bit or 10-bit MODE (the ADCRH register has been read but the ADCRL register has not). When blocking is active, the data transfer is blocked, COCO is not set, and the new result is lost. In the case of single conversions with the compare function enabled and the compare condition false, blocking has no effect and ADC operation is terminated. In all other cases of operation, when a data transfer is blocked, another conversion is initiated regardless of the state of ADCO (single or continuous conversions enabled).

If single conversions are enabled, the blocking mechanism could result in several discarded conversions and excess power consumption. To avoid this issue, the data registers must not be read after initiating a single conversion until the conversion completes.

### 15.4.4.3 Aborting Conversions

Any conversion in progress is aborted when:

- A write to ADCSC1 occurs (the current conversion is aborted and a new conversion is initiated, if ADCH are not all 1s).

- A write to ADCSC2, ADCCFG, ADCCVH, or ADCCVL occurs. This indicates a mode of operation change has occurred and the current conversion is therefore invalid.
- The MCU is reset.
- The MCU enters stop mode with ADACK not enabled.

When a conversion is aborted, the contents of the data registers, ADCRH and ADCRL, are not altered. However, they continue to be the values transferred after the completion of the last successful conversion. If the conversion was aborted by a reset, ADCRH and ADCRL return to their reset states.

#### 15.4.4.4 Power Control

The ADC module remains in its idle state until a conversion is initiated. If ADACK is selected as the conversion clock source, the ADACK clock generator is also enabled.

Power consumption when active can be reduced by setting ADLPC. This results in a lower maximum value for  $f_{ADCK}$  (see the electrical specifications).

#### 15.4.4.5 Sample Time and Total Conversion Time

The total conversion time depends on the sample time (as determined by ADLSMP), the MCU bus frequency, the conversion mode (8-bit, 10-bit or 12-bit), and the frequency of the conversion clock ( $f_{ADCK}$ ). After the module becomes active, sampling of the input begins. ADLSMP selects between short (3.5 ADCK cycles) and long (23.5 ADCK cycles) sample times. When sampling is complete, the converter is isolated from the input channel and a successive approximation algorithm is performed to determine the digital value of the analog signal. The result of the conversion is transferred to ADCRH and ADCRL upon completion of the conversion algorithm.

If the bus frequency is less than the  $f_{ADCK}$  frequency, precise sample time for continuous conversions cannot be guaranteed when short sample is enabled (ADLSMP=0). If the bus frequency is less than 1/11th of the  $f_{ADCK}$  frequency, precise sample time for continuous conversions cannot be guaranteed when long sample is enabled (ADLSMP=1).

The maximum total conversion time for different conditions is summarized in [Table 15-10](#).

**Table 15-10. Total Conversion Time vs. Control Conditions**

Conversion Type	ADICLK	ADLSMP	Max Total Conversion Time
Single or first continuous 8-bit	0x, 10	0	20 ADCK cycles + 5 bus clock cycles
Single or first continuous 10-bit or 12-bit	0x, 10	0	23 ADCK cycles + 5 bus clock cycles
Single or first continuous 8-bit	0x, 10	1	40 ADCK cycles + 5 bus clock cycles
Single or first continuous 10-bit or 12-bit	0x, 10	1	43 ADCK cycles + 5 bus clock cycles
Single or first continuous 8-bit	11	0	5 $\mu$ s + 20 ADCK + 5 bus clock cycles
Single or first continuous 10-bit or 12-bit	11	0	5 $\mu$ s + 23 ADCK + 5 bus clock cycles
Single or first continuous 8-bit	11	1	5 $\mu$ s + 40 ADCK + 5 bus clock cycles
Single or first continuous 10-bit or 12-bit	11	1	5 $\mu$ s + 43 ADCK + 5 bus clock cycles
Subsequent continuous 8-bit; $f_{BUS} \geq f_{ADCK}$	xx	0	17 ADCK cycles

**Table 15-10. Total Conversion Time vs. Control Conditions**

Conversion Type	ADICLK	ADLSMP	Max Total Conversion Time
Subsequent continuous 10-bit or 12-bit; $f_{BUS} \geq f_{ADCK}$	xx	0	20 ADCK cycles
Subsequent continuous 8-bit; $f_{BUS} \geq f_{ADCK}/11$	xx	1	37 ADCK cycles
Subsequent continuous 10-bit or 12-bit; $f_{BUS} \geq f_{ADCK}/11$	xx	1	40 ADCK cycles

The maximum total conversion time is determined by the clock source chosen and the divide ratio selected. The clock source is selectable by the ADICLK bits, and the divide ratio is specified by the ADIV bits. For example, in 10-bit mode, with the bus clock selected as the input clock source, the input clock divide-by-1 ratio selected, and a bus frequency of 8 MHz, then the conversion time for a single conversion is:

$$\text{Conversion time} = \frac{23 \text{ ADCK Cyc}}{8 \text{ MHz/1}} + \frac{5 \text{ bus Cyc}}{8 \text{ MHz}} = 3.5 \mu\text{s}$$

$$\text{Number of bus cycles} = 3.5 \mu\text{s} \times 8 \text{ MHz} = 28 \text{ cycles}$$

#### NOTE

The ADCK frequency must be between  $f_{ADCK}$  minimum and  $f_{ADCK}$  maximum to meet ADC specifications.

### 15.4.5 Automatic Compare Function

The compare function can be configured to check for an upper or lower limit. After the input is sampled and converted, the result is added to the two's complement of the compare value (ADCCVH and ADCCVL). When comparing to an upper limit (ACFGT = 1), if the result is greater-than or equal-to the compare value, COCO is set. When comparing to a lower limit (ACFGT = 0), if the result is less than the compare value, COCO is set. The value generated by the addition of the conversion result and the two's complement of the compare value is transferred to ADCRH and ADCRL.

Upon completion of a conversion while the compare function is enabled, if the compare condition is not true, COCO is not set and no data is transferred to the result registers. An ADC interrupt is generated upon the setting of COCO if the ADC interrupt is enabled (AIEN = 1).

#### NOTE

The compare function can monitor the voltage on a channel while the MCU is in wait or stop3 mode. The ADC interrupt wakes the MCU when the compare condition is met.

### 15.4.6 MCU Wait Mode Operation

Wait mode is a lower power-consumption standby mode from which recovery is fast because the clock sources remain active. If a conversion is in progress when the MCU enters wait mode, it continues until

completion. Conversions can be initiated while the MCU is in wait mode by means of the hardware trigger or if continuous conversions are enabled.

The bus clock, bus clock divided by two, and ADACK are available as conversion clock sources while in wait mode. The use of ALTCLK as the conversion clock source in wait is dependent on the definition of ALTCLK for this MCU. Consult the module introduction for information on ALTCLK specific to this MCU.

A conversion complete event sets the COCO and generates an ADC interrupt to wake the MCU from wait mode if the ADC interrupt is enabled (AIEN = 1).

## 15.4.7 MCU Stop3 Mode Operation

Stop mode is a low power-consumption standby mode during which most or all clock sources on the MCU are disabled.

### 15.4.7.1 Stop3 Mode With ADACK Disabled

If the asynchronous clock, ADACK, is not selected as the conversion clock, executing a stop instruction aborts the current conversion and places the ADC in its idle state. The contents of ADCRH and ADCRL are unaffected by stop3 mode. After exiting from stop3 mode, a software or hardware trigger is required to resume conversions.

### 15.4.7.2 Stop3 Mode With ADACK Enabled

If ADACK is selected as the conversion clock, the ADC continues operation during stop3 mode. For guaranteed ADC operation, the MCU's voltage regulator must remain active during stop3 mode. Consult the module introduction for configuration information for this MCU.

If a conversion is in progress when the MCU enters stop3 mode, it continues until completion. Conversions can be initiated while the MCU is in stop3 mode by means of the hardware trigger or if continuous conversions are enabled.

A conversion complete event sets the COCO and generates an ADC interrupt to wake the MCU from stop3 mode if the ADC interrupt is enabled (AIEN = 1).

#### NOTE

The ADC module can wake the system from low-power stop and cause the MCU to begin consuming run-level currents without generating a system level interrupt. To prevent this scenario, software should ensure the data transfer blocking mechanism (discussed in [Section 15.4.4.2, “Completing Conversions](#)) is cleared when entering stop3 and continuing ADC conversions.

## 15.4.8 MCU Stop2 Mode Operation

The ADC module is automatically disabled when the MCU enters stop2 mode. All module registers contain their reset values following exit from stop2. Therefore, the module must be re-enabled and re-configured following exit from stop2.

## 15.5 Initialization Information

This section gives an example that provides some basic direction on how to initialize and configure the ADC module. You can configure the module for 8-, 10-, or 12-bit resolution, single or continuous conversion, and a polled or interrupt approach, among many other options. Refer to [Table 15-7](#), [Table 15-8](#), and [Table 15-9](#) for information used in this example.

### NOTE

Hexadecimal values designated by a preceding 0x, binary values designated by a preceding %, and decimal values have no preceding character.

## 15.5.1 ADC Module Initialization Example

### 15.5.1.1 Initialization Sequence

Before the ADC module can be used to complete conversions, an initialization procedure must be performed. A typical sequence is as follows:

1. Update the configuration register (ADCCFG) to select the input clock source and the divide ratio used to generate the internal clock, ADCK. This register is also used for selecting sample time and low-power configuration.
2. Update status and control register 2 (ADCSC2) to select the conversion trigger (hardware or software) and compare function options, if enabled.
3. Update status and control register 1 (ADCSC1) to select whether conversions is continuous or completed only once, and to enable or disable conversion complete interrupts. The input channel on which conversions are performed is also selected here.

### 15.5.1.2 Pseudo-Code Example

In this example, the ADC module is set up with interrupts enabled to perform a single 10-bit conversion at low power with a long sample time on input channel 1, where the internal ADCK clock is derived from the bus clock divided by 1.

#### ADCCFG = 0x98 (%10011000)

Bit 7	ADLPC	1	Configures for low power (lowers maximum clock speed)
Bit 6:5	ADIV	00	Sets the ADCK to the input clock $\div 1$
Bit 4	ADLSMP	1	Configures for long sample time
Bit 3:2	MODE	10	Sets mode at 10-bit conversions
Bit 1:0	ADICLK	00	Selects bus clock as input clock source

#### ADCSC2 = 0x00 (%00000000)

Bit 7	ADACT	0	Flag indicates if a conversion is in progress
-------	-------	---	---

## Analog-to-Digital Converter (ADC12)

Bit 6	ADTRG	0	Software trigger selected
Bit 5	ACFE	0	Compare function disabled
Bit 4	ACFGT	0	Not used in this example
Bit 3:2		00	Reserved, always reads zero
Bit 1:0		00	Reserved for Freescale's internal use; always write zero

## ADCSC1 = 0x41 (%01000001)

Bit 7	COCO	0	Read-only flag set when a conversion completes
Bit 6	AIEN	1	Conversion complete interrupt enabled
Bit 5	ADCO	0	One conversion only (continuous conversions disabled)
Bit 4:0	ADCH	00001	Input channel 1 selected as ADC input channel

## ADCRH/L = 0xxx

Holds results of conversion. Read high byte (ADCRH) before low byte (ADCRL) so that conversion data cannot be overwritten with data from the next conversion.

## ADCCVH/L = 0xxx

Holds compare value when compare function enabled

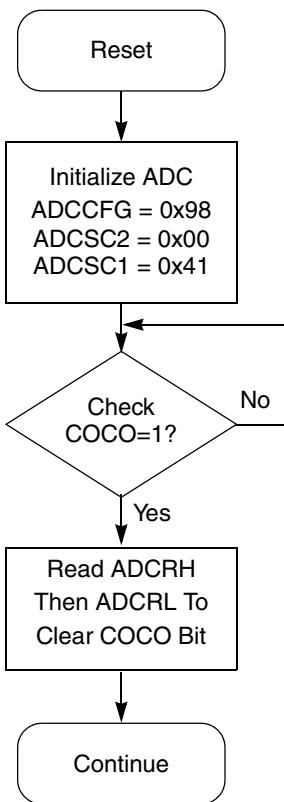


Figure 15-9. Initialization Flowchart for Example

## 15.6 Application Information

This section contains information for using the ADC module in applications. The ADC has been designed to be integrated into a microcontroller for use in embedded control applications requiring an A/D converter.

### 15.6.1 External Pins and Routing

The following sections discuss the external pins associated with the ADC module and how they should be used for best results.

#### 15.6.1.1 Analog Supply Pins

The ADC module has analog power and ground supplies ( $V_{DDAD}$  and  $V_{SSAD}$ ) available as separate pins on some devices.  $V_{SSAD}$  is shared on the same pin as the MCU digital  $V_{SS}$  on some devices. On other devices,  $V_{SSAD}$  and  $V_{DDAD}$  are shared with the MCU digital supply pins. In these cases, there are separate pads for the analog supplies bonded to the same pin as the corresponding digital supply so that some degree of isolation between the supplies is maintained.

When available on a separate pin,  $V_{DDAD}$  and  $V_{SSAD}$  must be connected to the same voltage potential as their corresponding MCU digital supply ( $V_{DD}$  and  $V_{SS}$ ) and must be routed carefully for maximum noise immunity and bypass capacitors placed as near as possible to the package.

If separate power supplies are used for analog and digital power, the ground connection between these supplies must be at the  $V_{SSAD}$  pin. This should be the only ground connection between these supplies if possible. The  $V_{SSAD}$  pin makes a good single point ground location.

#### 15.6.1.2 Analog Reference Pins

In addition to the analog supplies, the ADC module has connections for two reference voltage inputs. The high reference is  $V_{REFH}$ , which may be shared on the same pin as  $V_{DDAD}$  on some devices. The low reference is  $V_{REFL}$ , which may be shared on the same pin as  $V_{SSAD}$  on some devices.

When available on a separate pin,  $V_{REFH}$  may be connected to the same potential as  $V_{DDAD}$ , or may be driven by an external source between the minimum  $V_{DDAD}$  spec and the  $V_{DDAD}$  potential ( $V_{REFH}$  must never exceed  $V_{DDAD}$ ). When available on a separate pin,  $V_{REFL}$  must be connected to the same voltage potential as  $V_{SSAD}$ .  $V_{REFH}$  and  $V_{REFL}$  must be routed carefully for maximum noise immunity and bypass capacitors placed as near as possible to the package.

AC current in the form of current spikes required to supply charge to the capacitor array at each successive approximation step is drawn through the  $V_{REFH}$  and  $V_{REFL}$  loop. The best external component to meet this current demand is a  $0.1 \mu F$  capacitor with good high frequency characteristics. This capacitor is connected between  $V_{REFH}$  and  $V_{REFL}$  and must be placed as near as possible to the package pins. Resistance in the path is not recommended because the current causes a voltage drop that could result in conversion errors. Inductance in this path must be minimum (parasitic only).

### 15.6.1.3 Analog Input Pins

The external analog inputs are typically shared with digital I/O pins on MCU devices. The pin I/O control is disabled by setting the appropriate control bit in one of the pin control registers. Conversions can be performed on inputs without the associated pin control register bit set. It is recommended that the pin control register bit always be set when using a pin as an analog input. This avoids problems with contention because the output buffer is in its high impedance state and the pullup is disabled. Also, the input buffer draws DC current when its input is not at V<sub>DD</sub> or V<sub>SS</sub>. Setting the pin control register bits for all pins used as analog inputs should be done to achieve lowest operating current.

Empirical data shows that capacitors on the analog inputs improve performance in the presence of noise or when the source impedance is high. Use of 0.01  $\mu$ F capacitors with good high-frequency characteristics is sufficient. These capacitors are not necessary in all cases, but when used they must be placed as near as possible to the package pins and be referenced to V<sub>SSA</sub>.

For proper conversion, the input voltage must fall between V<sub>REFH</sub> and V<sub>REFL</sub>. If the input is equal to or exceeds V<sub>REFH</sub>, the converter circuit converts the signal to 0xFFFF (full scale 12-bit representation), 0x3FF (full scale 10-bit representation) or 0xFF (full scale 8-bit representation). If the input is equal to or less than V<sub>REFL</sub>, the converter circuit converts it to 0x000. Input voltages between V<sub>REFH</sub> and V<sub>REFL</sub> are straight-line linear conversions. There is a brief current associated with V<sub>REFL</sub> when the sampling capacitor is charging. The input is sampled for 3.5 cycles of the ADCK source when ADLSMP is low, or 23.5 cycles when ADLSMP is high.

For minimal loss of accuracy due to current injection, pins adjacent to the analog input pins should not be transitioning during conversions.

## 15.6.2 Sources of Error

Several sources of error exist for A/D conversions. These are discussed in the following sections.

### 15.6.2.1 Sampling Error

For proper conversions, the input must be sampled long enough to achieve the proper accuracy. Given the maximum input resistance of approximately 7k $\Omega$  and input capacitance of approximately 5.5 pF, sampling to within 1/4LSB (at 12-bit resolution) can be achieved within the minimum sample window (3.5 cycles @ 8 MHz maximum ADCK frequency) provided the resistance of the external analog source (R<sub>AS</sub>) is kept below 2 k $\Omega$ .

Higher source resistances or higher-accuracy sampling is possible by setting ADLSMP (to increase the sample window to 23.5 cycles) or decreasing ADCK frequency to increase sample time.

### 15.6.2.2 Pin Leakage Error

Leakage on the I/O pins can cause conversion error if the external analog source resistance (R<sub>AS</sub>) is high. If this error cannot be tolerated by the application, keep R<sub>AS</sub> lower than V<sub>DDAD</sub> / (2<sup>N</sup>\*I<sub>LEAK</sub>) for less than 1/4LSB leakage error (N = 8 in 8-bit, 10 in 10-bit or 12 in 12-bit mode).

### 15.6.2.3 Noise-Induced Errors

System noise that occurs during the sample or conversion process can affect the accuracy of the conversion. The ADC accuracy numbers are guaranteed as specified only if the following conditions are met:

- There is a 0.1  $\mu\text{F}$  low-ESR capacitor from  $V_{\text{REFH}}$  to  $V_{\text{REFL}}$ .
- There is a 0.1  $\mu\text{F}$  low-ESR capacitor from  $V_{\text{DDAD}}$  to  $V_{\text{SSAD}}$ .
- If inductive isolation is used from the primary supply, an additional 1  $\mu\text{F}$  capacitor is placed from  $V_{\text{DDAD}}$  to  $V_{\text{SSAD}}$ .
- $V_{\text{SSAD}}$  (and  $V_{\text{REFL}}$ , if connected) is connected to  $V_{\text{SS}}$  at a quiet point in the ground plane.
- Operate the MCU in wait or stop3 mode before initiating (hardware triggered conversions) or immediately after initiating (hardware or software triggered conversions) the ADC conversion.
  - For software triggered conversions, immediately follow the write to ADCSC1 with a stop instruction.
  - For stop3 mode operation, select ADACK as the clock source. Operation in stop3 reduces  $V_{\text{DD}}$  noise but increases effective conversion time due to stop recovery.
- There is no I/O switching, input or output, on the MCU during the conversion.

There are some situations where external system activity causes radiated or conducted noise emissions or excessive  $V_{\text{DD}}$  noise is coupled into the ADC. In these situations, or when the MCU cannot be placed in wait or stop3 or I/O activity cannot be halted, these recommended actions may reduce the effect of noise on the accuracy:

- Place a 0.01  $\mu\text{F}$  capacitor ( $C_{\text{AS}}$ ) on the selected input channel to  $V_{\text{REFL}}$  or  $V_{\text{SSAD}}$  (this improves noise issues, but affects the sample rate based on the external analog source resistance).
- Average the result by converting the analog input many times in succession and dividing the sum of the results. Four samples are required to eliminate the effect of a 1LSB, one-time error.
- Reduce the effect of synchronous noise by operating off the asynchronous clock (ADACK) and averaging. Noise synchronous to ADCK cannot be averaged out.

### 15.6.2.4 Code Width and Quantization Error

The ADC quantizes the ideal straight-line transfer function into 4096 steps (in 12-bit mode). Each step ideally has the same height (1 code) and width. The width is defined as the delta between the transition points to one code and the next. The ideal code width for an N bit converter (in this case N can be 8, 10 or 12), defined as 1LSB, is:

$$1 \text{ lsb} = (V_{\text{REFH}} - V_{\text{REFL}}) / 2^N$$

*Eqn. 15-2*

There is an inherent quantization error due to the digitization of the result. For 8-bit or 10-bit conversions the code transitions when the voltage is at the midpoint between the points where the straight line transfer function is exactly represented by the actual transfer function. Therefore, the quantization error is  $\pm 1/2$  lsb in 8- or 10-bit mode. As a consequence, however, the code width of the first (0x000) conversion is only 1/2 lsb and the code width of the last (0xFF or 0x3FF) is 1.5 lsb.

For 12-bit conversions the code transitions only after the full code width is present, so the quantization error is -1 lsb to 0 lsb and the code width of each step is 1 lsb.

### 15.6.2.5 Linearity Errors

The ADC may also exhibit non-linearity of several forms. Every effort has been made to reduce these errors but the system should be aware of them because they affect overall accuracy. These errors are:

- Zero-scale error ( $E_{ZS}$ ) (sometimes called offset) — This error is defined as the difference between the actual code width of the first conversion and the ideal code width (1/2 lsb in 8-bit or 10-bit modes and 1 lsb in 12-bit mode). If the first conversion is 0x001, the difference between the actual 0x001 code width and its ideal (1 lsb) is used.
- Full-scale error ( $E_{FS}$ ) — This error is defined as the difference between the actual code width of the last conversion and the ideal code width (1.5 lsb in 8-bit or 10-bit modes and 1LSB in 12-bit mode). If the last conversion is 0x3FE, the difference between the actual 0x3FE code width and its ideal (1LSB) is used.
- Differential non-linearity (DNL) — This error is defined as the worst-case difference between the actual code width and the ideal code width for all conversions.
- Integral non-linearity (INL) — This error is defined as the highest-value the (absolute value of the) running sum of DNL achieves. More simply, this is the worst-case difference of the actual transition voltage to a given code and its corresponding ideal transition voltage, for all codes.
- Total unadjusted error (TUE) — This error is defined as the difference between the actual transfer function and the ideal straight-line transfer function and includes all forms of error.

### 15.6.2.6 Code Jitter, Non-Monotonicity, and Missing Codes

Analog-to-digital converters are susceptible to three special forms of error. These are code jitter, non-monotonicity, and missing codes.

Code jitter is when, at certain points, a given input voltage converts to one of two values when sampled repeatedly. Ideally, when the input voltage is infinitesimally smaller than the transition voltage, the converter yields the lower code (and vice-versa). However, even small amounts of system noise can cause the converter to be indeterminate (between two codes) for a range of input voltages around the transition voltage. This range is normally around  $\pm 1/2$  lsb in 8-bit or 10-bit mode, or around 2 lsb in 12-bit mode, and increases with noise.

This error may be reduced by repeatedly sampling the input and averaging the result. Additionally the techniques discussed in [Section 15.6.2.3](#) reduces this error.

Non-monotonicity is defined as when, except for code jitter, the converter converts to a lower code for a higher input voltage. Missing codes are those values never converted for any input value.

In 8-bit or 10-bit mode, the ADC is guaranteed to be monotonic and have no missing codes.

# Chapter 16

## Fast Ethernet Controller (FEC)

### 16.1 Introduction

This chapter provides a feature-set overview, a functional block diagram, and transceiver connection information for the 10 and 100 Mbps MII (media independent interface), as well as the 7-wire serial interface. Additionally, detailed descriptions of operation and the programming model are included.

#### NOTE

- Use pin mux control registers from [Section 2.3, “Pin Mux Controls”](#) to assign FEC signals to the MCF51CN128 package pins.
- Most pin functions default to GPIO and must be software configured before using FEC.

#### 16.1.1 Overview

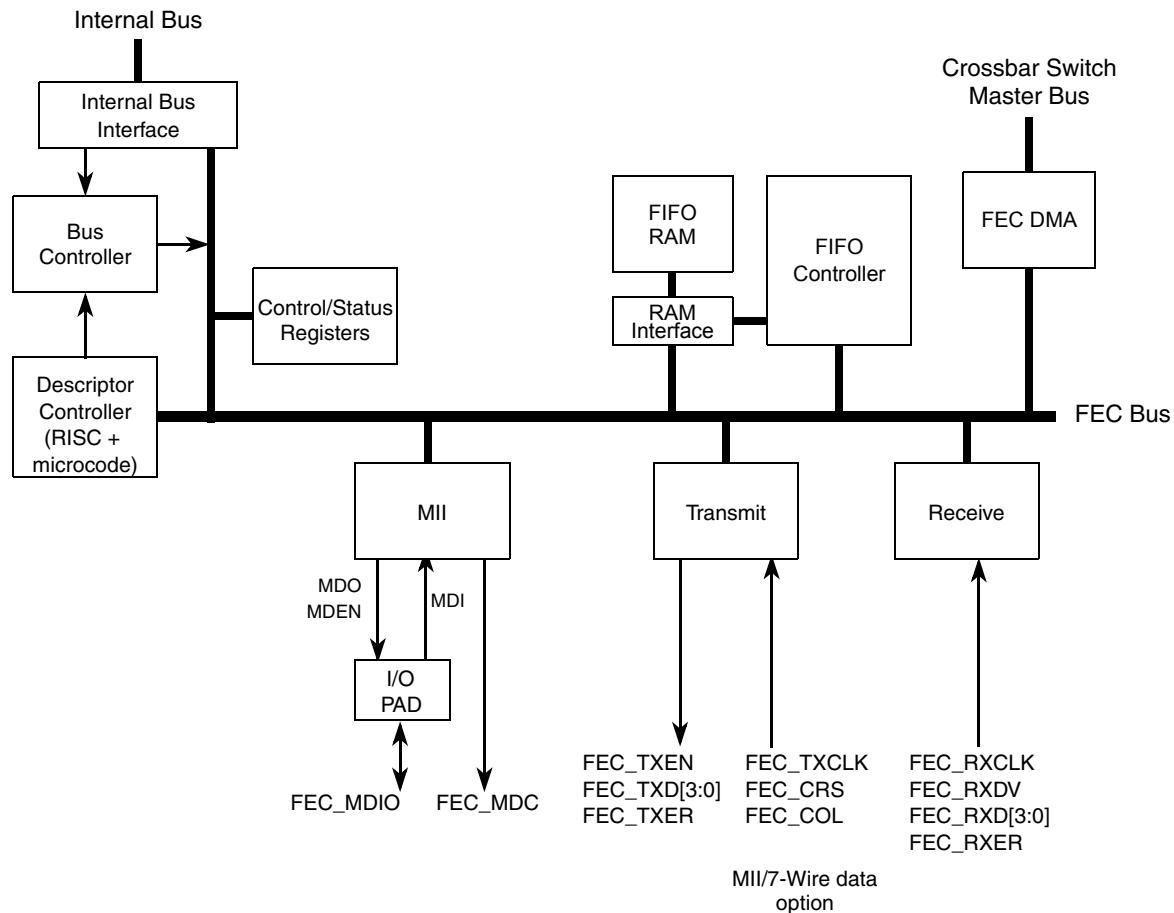
The Ethernet media access controller (MAC) supports 10 and 100 Mbps Ethernet/IEEE 802.3 networks. An external transceiver interface and transceiver function are required to complete the interface to the media. The FEC supports three different standard MAC-PHY (physical) interfaces for connection to an external Ethernet transceiver. The FEC supports the 10/100 Mbps MII and the 10 Mbps-only 7-wire interface.

#### NOTE

The pin multiplexing and control module must be configured to enable the peripheral function of the appropriate pins (refer to [Chapter 9, “Parallel Input/Output Control”](#)) prior to configuring the FEC.

## 16.1.2 Block Diagram

Figure 16-1 shows the block diagram of the FEC. The FEC is implemented with a combination of hardware and microcode. The off-chip (Ethernet) interfaces are compliant with industry and IEEE 802.3 standards.



**Figure 16-1. FEC Block Diagram**

The descriptor controller is a RISC-based controller providing these functions in the FEC:

- Initialization (those internal registers not initialized by you 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

The RAM is the focal point of all data flow in the Fast Ethernet controller and divides into transmit and receive FIFOs. The FIFO boundaries are programmable using the FRSR register. User data flows to/from the DMA block from/to the receive/transmit FIFOs. Transmit data flows from the transmit FIFO into the transmit block, and receive data flows from the receive block into the receive FIFO.

You control the FEC by writing into control registers located in each block. The CSR (control and status registers) block provides global control (Ethernet reset and enable) and interrupt managing 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\_MDC (management data clock) and FEC\_MDIO (management data input/output) lines of the MII interface.

The FEC 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).

### 16.1.3 Features

The FEC incorporates the following features:

- Support for three different Ethernet physical interfaces:
  - 100-Mbps IEEE 802.3 MII
  - 10-Mbps IEEE 802.3 MII
  - 10-Mbps 7-wire interface (industry standard)
- IEEE 802.3 full duplex flow control
- Programmable max frame length supports IEEE 802.1 VLAN tags and priority
- Support for full-duplex operation (200 Mbps throughput) with a minimum internal bus clock rate of 50 MHz
- Support for half-duplex operation (100 Mbps throughput) with a minimum internal bus clock rate of 50 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

## 16.2 Modes of Operation

The primary operational modes are described in this section.

## 16.2.1 Full and Half Duplex Operation

Full duplex mode is for use on point-to-point links between switches or end node to switch. Half duplex mode works in connections between an end node and a repeater or between repeaters. TCR[FDEN] controls duplex mode selection.

When configured for full duplex mode, flow control may be enabled. Refer to the TCR[RFC\_PAUSE,TFC\_PAUSE] bits, the RCR[FCE] bit, and [Section 16.5.11, “Full Duplex Flow Control”](#) for more details.

## 16.2.2 Interface Options

The following interface options are supported. A detailed discussion of the interface configurations is provided in [Section 16.5.6, “Network Interface Options.”](#)

### 16.2.2.1 10 Mbps and 100 Mbps MII Interface

The IEEE 802.3 standard defines the media independent interface (MII) for 10/100 Mbps operation. The MAC-PHY interface may be configured to operate in MII mode by setting RCR[MII\_MODE].

FEC\_TXCLK and FEC\_RXCLK pins driven by the external transceiver determine the operation speed. The transceiver auto-negotiates the speed or software controls it via the serial management interface (FEC\_MDC/FEC\_MDIO pins) to the transceiver. Refer to the MMFR and MSCR register descriptions, as well as the section on the MII, for a description of how to read and write registers in the transceiver via this interface.

### 16.2.2.2 10 Mbps 7-Wire Interface Operation

The FEC supports 7-wire interface used by many 10 Mbps Ethernet transceivers. The RCR[MII\_MODE] bit controls this functionality. If this bit is cleared, MII mode is disabled and the 10 Mbps 7-wire mode is enabled.

## 16.2.3 Address Recognition Options

The address options supported are promiscuous, broadcast reject, individual address (hash or exact match), and multicast hash match. Address recognition options are discussed in detail in [Section 16.5.9, “Ethernet Address Recognition.”](#)

## 16.2.4 Internal Loopback

Internal loopback mode is selected via RCR[LOOP]. Loopback mode is discussed in detail in [Section 16.5.14, “MII Internal and External Loopback.”](#)

## 16.3 External Signal Description

[Table 16-1](#) describes the various FEC signals, as well as indicating which signals work in available modes.

**Table 16-1. FEC Signal Descriptions**

<b>Signal Name</b>	<b>MII</b>	<b>7-wire</b>	<b>Description</b>
FEC_COL	X	X	Asserted upon detection of a collision and remains asserted while the collision persists. This signal is not defined for full-duplex mode.
FEC_CRS	X	—	When asserted, indicates that transmit or receive medium is not idle.
FEC_MDC	X	—	Output clock which provides a timing reference to the PHY for data transfers on the FEC_MDIO signal.
FEC_MDIO	X	—	Transfers control information between the external PHY and the media-access controller. Data is synchronous to FEC_MDC. 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.
FEC_RXCLK	X	X	Provides a timing reference for FEC_RXDV, FEC_RXD[3:0], and FEC_RXER.
FEC_RXDV	X	X	Asserting the FEC_RXDV input indicates that the PHY has valid nibbles present on the MII. FEC_RXDV should remain asserted from the first recovered nibble of the frame through to the last nibble. Assertion of FEC_RXDV must start no later than the SFD and exclude any EOF.
FEC_RXD0	X	X	This pin contains the Ethernet input data transferred from the PHY to the media-access controller when FEC_RXDV is asserted.
FEC_RXD1	X	—	This pin contains the Ethernet input data transferred from the PHY to the media access controller when FEC_RXDV is asserted.
FEC_RXD[3:2]	X	—	These pins contain the Ethernet input data transferred from the PHY to the media access controller when FEC_RXDV is asserted.
FEC_RXER	X	—	When asserted with FEC_RXDV, indicates that the PHY has detected an error in the current frame. When FEC_RXDV is not asserted FEC_RXER has no effect.
FEC_TXCLK	X	X	Input clock which provides a timing reference for FEC_TXEN, FEC_TXD[3:0] and FEC_TXER.
FEC_TXD0	X	X	The serial output Ethernet data and is only valid during the assertion of FEC_TXEN.
FEC_TXD1	X	—	This pin contains the serial output Ethernet data and is valid only during assertion of FEC_TXEN.
FEC_TXD[3:2]	X	—	These pins contain the serial output Ethernet data and are valid only during assertion of FEC_TXEN.
FEC_TXEN	X	X	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_TXCLK following the final nibble of the frame.
FEC_TXER	X	—	When asserted for one or more clock cycles while FEC_TXEN is also asserted, the PHY sends one or more illegal symbols. FEC_TXER has no effect at 10 Mbps or when FEC_TXEN is negated.

## 16.4 Memory Map/Register Definition

The FEC is programmed by a combination of control/status registers (CSRs) and buffer descriptors. The CSRs control operation modes and extract global status information. The descriptors pass data buffers and related buffer information between the hardware and software.

Table 16-2 shows the FEC register memory map.

**Table 16-2. FEC Register Memory Map**

Offset	Register	Width (bits)	Access	Reset Value	Section/Page
0x004	Interrupt Event Register (EIR)	32	R/W	0x0000_0000	<a href="#">16.4.1/16-6</a>
0x008	Interrupt Mask Register (EIMR)	32	R/W	0x0000_0000	<a href="#">16.4.2/16-8</a>
0x010	Receive Descriptor Active Register (RDAR)	32	R/W	0x0000_0000	<a href="#">16.4.3/16-8</a>
0x014	Transmit Descriptor Active Register (TDAR)	32	R/W	0x0000_0000	<a href="#">16.4.4/16-9</a>
0x024	Ethernet Control Register (ECR)	32	R/W	0xF000_0000	<a href="#">16.4.5/16-10</a>
0x040	MII Management Frame Register (MMFR)	32	R/W	Undefined	<a href="#">16.4.6/16-10</a>
0x044	MII Speed Control Register (MSCR)	32	R/W	0x0000_0000	<a href="#">16.4.7/16-12</a>
0x084	Receive Control Register (RCR)	32	R/W	0x05EE_0001	<a href="#">16.4.8/16-13</a>
0x0C4	Transmit Control Register (TCR)	32	R/W	0x0000_0000	<a href="#">16.4.9/16-14</a>
0x0E4	Physical Address Low Register (PALR)	32	R/W	Undefined	<a href="#">16.4.10/16-15</a>
0x0E8	Physical Address High Register (PAUR)	32	R/W	See Section	<a href="#">16.4.11/16-15</a>
0x0EC	Opcode/Pause Duration (OPD)	32	R/W	See Section	<a href="#">16.4.12/16-16</a>
0x118	Descriptor Individual Upper Address Register (IAUR)	32	R/W	Undefined	<a href="#">16.4.13/16-16</a>
0x11C	Descriptor Individual Lower Address Register (IALR)	32	R/W	Undefined	<a href="#">16.4.14/16-17</a>
0x120	Descriptor Group Upper Address Register (GAUR)	32	R/W	Undefined	<a href="#">16.4.15/16-17</a>
0x124	Descriptor Group Lower Address Register (GALR)	32	R/W	Undefined	<a href="#">16.4.16/16-18</a>
0x144	Transmit FIFO Watermark (TFWR)	32	R/W	0x0100_0001	<a href="#">16.4.17/16-18</a>
0x14C	FIFO Receive Bound Register (FRBR)	32	R	0x0000_0600	<a href="#">16.4.18/16-19</a>
0x150	FIFO Receive FIFO Start Register (FRSR)	32	R	0x0000_0500	<a href="#">16.4.19/16-19</a>
0x180	Pointer to Receive Descriptor Ring (ERDSR)	32	R/W	Undefined	<a href="#">16.4.20/16-20</a>
0x184	Pointer to Transmit Descriptor Ring (ETDSR)	32	R/W	Undefined	<a href="#">16.4.21/16-20</a>
0x188	Maximum Receive Buffer Size (EMRBR)	32	R/W	Undefined	<a href="#">16.4.22/16-21</a>

## 16.4.1 Ethernet Interrupt Event Register (EIR)

When an event occurs that sets a bit in EIR, an interrupt occurs if the corresponding bit in the interrupt mask register (EIMR) is also set. Writing a 1 to an EIR bit clears it; writing 0 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.

Offset: 0x004

Access: User read/write

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

Figure 16-2. Ethernet Interrupt Event Register (EIR)

Table 16-3. EIR Field Descriptions

Field	Description
31 HBERR	Heartbeat error. Indicates TCR[HBC] is set and that the COL input was not asserted within the heartbeat window following a transmission.
30 BABR	Babbling receive error. Indicates a frame was received with length in excess of RCR[MAX_FL] bytes.
29 BABT	Babbling transmit error. Indicates the transmitted frame length exceeds RCR[MAX_FL] bytes. Usually this condition is caused by a frame that is too long is placed into the transmit data buffer(s). Truncation does not occur.
28 GRA	Graceful stop complete. Indicates the graceful stop is complete. During graceful stop the transmitter is placed into a pause state after completion of the frame currently being transmitted. This bit is set by one of three conditions: 1) A graceful stop initiated by the setting of the TCR[GTS] bit is now complete. 2) A graceful stop initiated by the setting of the TCR[TFC_PAUSE] bit is now complete. 3) A graceful stop initiated by the reception of a valid full duplex flow control pause frame is now complete. Refer to <a href="#">Section 16.5.11, “Full Duplex Flow Control”</a> .
27 TXF	Transmit frame interrupt. Indicates a frame has been transmitted and the last corresponding buffer descriptor has been updated.
26 TXB	Transmit buffer interrupt. Indicates a transmit buffer descriptor has been updated.
25 RXF	Receive frame interrupt. Indicates a frame has been received and the last corresponding buffer descriptor has been updated.
24 RXB	Receive buffer interrupt. Indicates a receive buffer descriptor not the last in the frame has been updated.
23 MII	MII interrupt. Indicates the MII has completed the data transfer requested.
22 EBERR	Ethernet bus error. Indicates a system bus error occurred when a DMA transaction is underway. When the EBERR bit is set, ECR[ETHER_EN] is cleared, halting frame processing by the FEC. When this occurs, software needs to ensure that the FIFO controller and DMA also soft reset.
21 LC	Late collision. Indicates a collision occurred beyond the collision window (slot time) in half duplex mode. The frame truncates with a bad CRC and the remainder of the frame is discarded.

**Table 16-3. EIR Field Descriptions (continued)**

Field	Description
20 RL	Collision retry limit. Indicates 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 commences. This error can only occur in half duplex mode.
19 UN	Transmit FIFO underrun. Indicates 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, must be cleared.

### 16.4.2 Interrupt Mask Register (EIMR)

The EIMR register controls which interrupt events are allowed to generate actual interrupts. All implemented bits in this CSR are read/write. A hardware reset clears this register. If the corresponding bits in the EIR and EIMR registers are set, an interrupt is generated. The interrupt signal remains asserted until a 1 is written to the EIR bit (write 1 to clear) or a 0 is written to the EIMR bit.

Offset: 0x008 Access: User read/write

	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

**Figure 16-3. Ethernet Interrupt Mask Register (EIMR)****Table 16-4. EIMR Field Descriptions**

Field	Description
31–19 See <a href="#">Figure 16-3</a> and <a href="#">Table 16-3</a>	Interrupt mask. Each bit corresponds to an interrupt source defined by the EIR register. The corresponding EIMR bit determines whether an interrupt condition can generate an interrupt. At every processor clock, the EIR samples the signal generated by the interrupting source. The corresponding EIR bit reflects the state of the interrupt signal even if the corresponding EIMR bit is set. 0 The corresponding interrupt source is masked. 1 The corresponding interrupt source is not masked.
18–0	Reserved, must be cleared.

### 16.4.3 Receive Descriptor Active Register (RDAR)

RDAR is a command register, written by the user, indicating the receive descriptor ring is updated (the driver produced empty receive buffers with the empty bit set).

When the register is written, the RDAR bit is set. This is independent of the data actually written by the user. When set, the FEC polls the receive descriptor ring and processes receive frames (provided

ECR[ETHER\_EN] is also set). After the FEC polls a receive descriptor whose empty bit is not set, FEC clears the RDAR bit and ceases receive descriptor ring polling until the user sets the bit again, signifying that additional descriptors are placed into the receive descriptor ring.

The RDAR register is cleared at reset and when ECR[ETHER\_EN] is cleared.

Offset: 0x010																Access: User read/write															
R																RDAR															
W																															
Reset																0 0															

Figure 16-4. Receive Descriptor Active Register (RDAR)

Table 16-5. RDAR Field Descriptions

Field	Description
31–25	Reserved, must be cleared.
24 RDAR	Set to 1 when this register is written, regardless of the value written. Cleared by the FEC device when no additional empty descriptors remain in the receive ring. Also cleared when ECR[ETHER_EN] is cleared.
23–0	Reserved, must be cleared.

#### 16.4.4 Transmit Descriptor Active Register (TDAR)

The TDAR is a command register which the user writes to indicate the transmit descriptor ring is updated (transmit buffers have been produced by the driver with the ready bit set in the buffer descriptor).

When 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 polls the transmit descriptor ring and processes transmit frames (provided ECR[ETHER\_EN] is also set). After the FEC polls a transmit descriptor that is a ready bit not set, FEC clears the TDAR bit and ceases transmit descriptor ring polling until the user sets the bit again, signifying additional descriptors are placed into the transmit descriptor ring.

The TDAR register is cleared at reset, when ECR[ETHER\_EN] is cleared, or when ECR[RESET] is set.

Offset: 0x014																Access: User read/write															
R																TDAR															
W																															
Reset																0 0															

Figure 16-5. Transmit Descriptor Active Register (TDAR)

Table 16-6. TDAR Field Descriptions

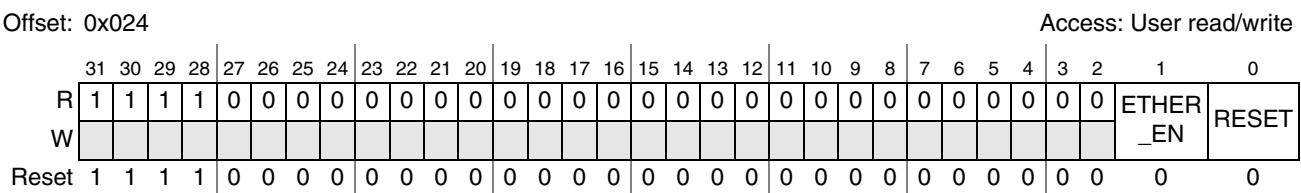
Field	Description
31–25	Reserved, must be cleared.

**Table 16-6. TDAR Field Descriptions (continued)**

Field	Description
24 TDAR	Set to 1 when this register is written, regardless of the value written. Cleared by the FEC device when no additional ready descriptors remain in the transmit ring. Also cleared when ECR[ETHER_EN] is cleared.
23–0	Reserved, must be cleared.

### **16.4.5 Ethernet Control Register (ECR)**

ECR is a read/write user register, though hardware may alter fields in this register as well. The ECR enables/disables the FEC.



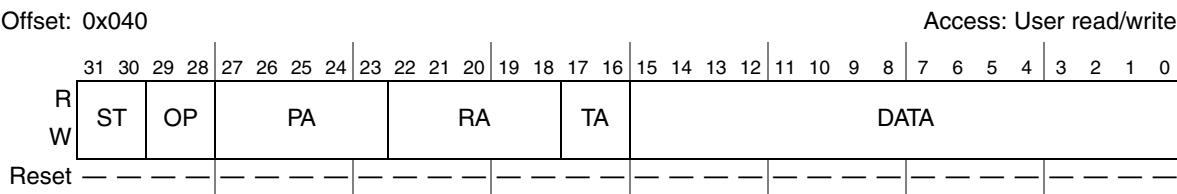
**Figure 16-6. Ethernet Control Register (ECR)**

**Table 16-7. ECR Field Descriptions**

Field	Description
31–2	Reserved, must be cleared.
1 ETHER_EN	<p>When this bit is set, FEC is enabled, and reception and transmission are possible. When this bit is cleared, reception immediately stops and transmission stops 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 cleared, the DMA, buffer descriptor, and FIFO control logic are reset, including the buffer descriptor and FIFO pointers. Hardware alters the ETHER_EN bit under the following conditions:</p> <ul style="list-style-type: none"> <li>• ECR[RESET] is set by software, in which case ETHER_EN is cleared</li> <li>• An error condition causes the EIR[EBERR] bit to set, in which case ETHER_EN is 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. ECR[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 eight internal bus clock cycles after this bit is set.

#### **16.4.6 MII Management Frame Register (MMFR)**

The MMFR is user-accessible and does not reset to a defined value. The MMFR register is 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 causes a management frame to be sourced unless the MSCR is programmed to 0. If MSCR is cleared while MMFR is written and then MSCR is written with a non-zero value, an MII frame is generated with the data previously written to the MMFR. This allows MMFR and MSCR to be programmed in either order if MSCR is currently zero.



**Figure 16-7. MII Management Frame Register (MMFR)**

**Table 16-8. MMFR Field Descriptions**

Field	Description
31–30 ST	Start of frame delimiter. These bits must be programmed to 0b01 for a valid MII management frame.
29–28 OP	Operation code. 00 Write frame operation, but not MII compliant.
	01 Write frame operation for a valid MII management frame.
	10 Read frame operation for a valid MII management frame.
	11 Read frame operation, but not 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, write the MMFR register. To generate a valid read or write management frame, ST field must be written with a 01 pattern, and the TA field must be written with a 10. If other patterns are written to these fields, a frame is generated, but does not comply with the IEEE 802.3 MII definition.

To generate an IEEE 802.3-compliant MII Management Interface write frame (write to a PHY register), the user must write {01 01 PHYAD REGAD 10 DATA} to the MMFR register. Writing this pattern causes the control logic to shift out the data in the MMFR register following a preamble generated by the control state machine. During this time, contents of the MMFR register are altered as the contents are serially shifted and are unpredictable if read by the user. After the write management frame operation completes, the MII interrupt is generated. At this time, contents of the MMFR register match the original value written.

To generate an MII management interface read frame (read a PHY register), the user must write {01 10 PHYAD REGAD 10 XXXX} to the MMFR register (the content of the DATA field is a don't care). Writing this pattern causes the control logic to shift out the data in the MMFR register following a preamble generated by the control state machine. During this time, contents of the MMFR register are altered as the contents are serially shifted and are unpredictable if read by the user. After the read management frame operation completes, the MII interrupt is generated. At this time, the contents of the MMFR register match

the original value written except for the DATA field whose contents are replaced by the value read from the PHY register.

If the MMFR register is written while frame generation is in progress, the frame contents are altered. Software must use the MII interrupt to avoid writing to the MMFR register while frame generation is in progress.

#### 16.4.7 MII Speed Control Register (MSCR)

The MSCR provides control of the MII clock (FEC\_MDC pin) frequency and allows a preamble drop on the MII management frame.

**Figure 16-8. MII Speed Control Register (MSCR)**

**Table 16-9. MSCR Field Descriptions**

Field	Description
31–8	Reserved, must be cleared.
7 DIS_PRE	Setting this bit causes the preamble (32 ones) 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	Controls the frequency of the MII management interface clock (FEC_MDC) relative to the internal bus clock. A value of 0 in this field turns off the FEC_MDC and leaves it in low voltage state. Any non-zero value results in the FEC_MDC frequency of $1/(MII\_SPEED \times 2)$ of the internal bus frequency.
0	Reserved, must be cleared.

The MII\_SPEED field must be programmed with a value to provide an FEC\_MDC frequency of less than or equal to 2.5 MHz to be compliant with the IEEE 802.3 MII specification. The MII\_SPEED must be set to a non-zero value to source a read or write management frame. After the management frame is complete, the MSCR register may optionally be set to 0 to turn off the FEC\_MDC. The FEC\_MDC generated has a 50% duty cycle except when MII\_SPEED changes during operation (change takes effect following a rising or falling edge of FEC\_MDC).

If the internal bus clock is 25 MHz, programming this register to 0x0000\_0005 results in an FEC\_MDC as stated the equation below.

$$25 \text{ MHz} \times \frac{1}{5 \times 2} = 2.5 \text{ MHz} \quad \text{Eqn. 16-1}$$

A table showing optimum values for MII\_SPEED as a function of internal bus clock frequency is provided below.

**Table 16-10. Programming Examples for MSCR**

Internal FEC Clock Frequency	MSCR[MII_SPEED]	FEC_MDC frequency
25 MHz	0x5	2.50 MHz
33 MHz	0x7	2.36 MHz
40 MHz	0x8	2.50 MHz
50 MHz	0xA	2.50 MHz
66 MHz	0xE	2.36 MHz

### 16.4.8 Receive Control Register (RCR)

RCR controls the operational mode of the receive block and must be written only when ECR[ETHER\_EN] is cleared (initialization time).

Access: User read/write																
R	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
W	0	0	0	0	0	MAX_FL										
Reset	0	0	0	0	0	1	0	1	1	1	1	0	1	1	1	0
R	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
W	0	0	0	0	0	0	0	0	0	0	0	0	PROM	MII_MODE	DRT	LOOP
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

**Figure 16-9. Receive Control Register (RCR)****Table 16-11. RCR Field Descriptions**

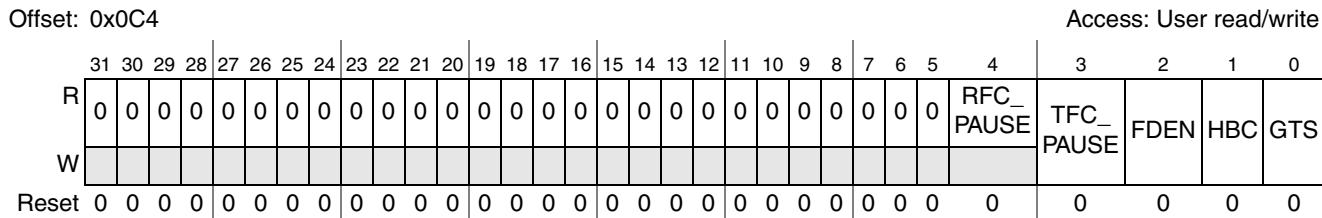
Field	Description
31–27	Reserved, must be cleared.
26–16 MAX_FL	Maximum frame length. Resets to decimal 1518. Length is measured starting at DA and includes the CRC at the end of the frame. Transmit frames longer than MAX_FL causes the BABT interrupt to occur. Receive frames longer than MAX_FL causes the BABR interrupt to occur and sets the LG bit in the end of frame receive buffer descriptor. The recommended default value to be programmed is 1518 or 1522 if VLAN tags are supported.
15–6	Reserved, must be cleared.
5 FCE	Flow control enable. If asserted, the receiver detects PAUSE frames. Upon PAUSE frame detection, the transmitter stops transmitting data frames for a given duration.
4 BC_REJ	Broadcast frame reject. If asserted, frames with DA (destination address) equal to FFFF_FFFF_FFFF are rejected unless the PROM bit is set. If BC_REJ and PROM are set, frames with broadcast DA are accepted and the M(MISS) is set in the receive buffer descriptor.
3 PROM	Promiscuous mode. All frames are accepted regardless of address matching.

**Table 16-11. RCR Field Descriptions (continued)**

Field	Description
2 MII_MODE	Media independent interface mode. Selects the external interface mode for transmit and receive blocks. 0 7-wire mode (used only for serial 10 Mbps) 1 MII mode
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 transmit output signals are not asserted. The internal bus clock substitutes for the FEC_TXCLK when LOOP is asserted. DRT must be set to 0 when setting LOOP.

### 16.4.9 Transmit Control Register (TCR)

TCR is read/write and configures the transmit block. This register is cleared at system reset. Bits 2 and 1 must be modified only when ECR[ETHER\_EN] is cleared.

**Figure 16-10. Transmit Control Register (TCR)****Table 16-12. TCR Field Descriptions**

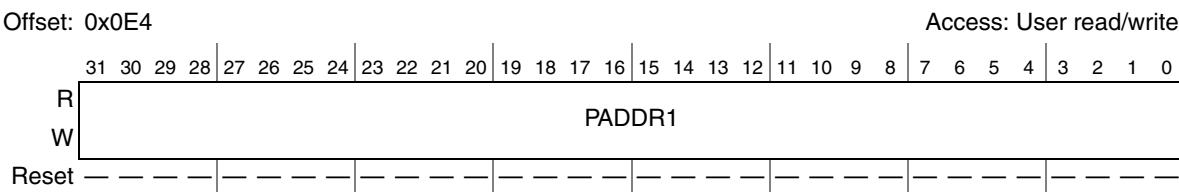
Field	Description
31–5	Reserved, must be cleared.
4 RFC_PAUSE	Receive frame control pause. This read-only status bit is asserted when a full duplex flow control pause frame is received and the transmitter pauses for the duration defined in this pause frame. This bit automatically clears when the pause duration is complete.
3 TFC_PAUSE	Transmit frame control pause. Transmits a PAUSE frame when asserted. When this bit is set, the MAC stops transmission of data frames after the current transmission is complete. At this time, GRA interrupt in the EIR register is asserted. With transmission of data frames stopped, MAC transmits a MAC Control PAUSE frame. Next, the MAC clears the TFC_PAUSE bit and resumes transmitting data frames. If the transmitter pauses due to user assertion of GTS or reception of a PAUSE frame, the MAC may continue transmitting a MAC Control PAUSE frame.
2 FDEN	Full duplex enable. If set, frames transmit independent of carrier sense and collision inputs. This bit should only be modified when ECR[ETHER_EN] is cleared.

**Table 16-12. TCR Field Descriptions (continued)**

Field	Description
1 HBC	Heartbeat control. If set, the heartbeat check performs following end of transmission and the HB bit in the status register is set if the collision input does not assert within the heartbeat window. This bit should only be modified when ECR[ETHER_EN] is cleared.
0 GTS	Graceful transmit stop. When this bit is set, MAC stops transmission after any frame currently transmitted is complete and GRA interrupt in the EIR register is asserted. If frame transmission is not currently underway, the GRA interrupt is asserted immediately. After transmission finishes, clear GTS to restart. The next frame in the transmit FIFO is then transmitted. If an early collision occurs during transmission when GTS is set, transmission stops after the collision. The frame is transmitted again after GTS is cleared. There may be old frames in the transmit FIFO that transmit when GTS is reasserted. To avoid this, clear ECR[ETHER_EN] following the GRA interrupt.

#### 16.4.10 Physical Address Lower Register (PALR)

PALR 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 you must initialize it.



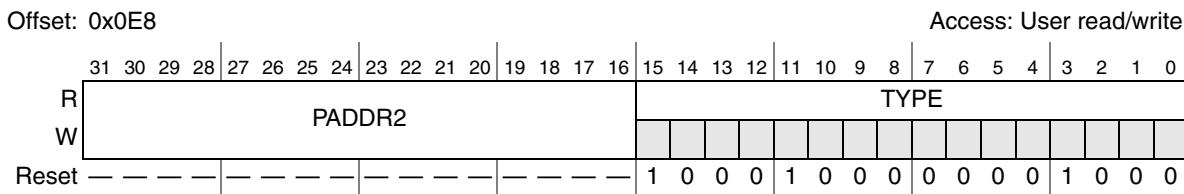
**Figure 16-11. Physical Address Lower Register (PALR)**

**Table 16-13. PALR Field Descriptions**

Field	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 are used for exact match and the source address field in PAUSE frames.

#### 16.4.11 Physical Address Upper Register (PAUR)

PAUR 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 contain a constant type field (0x8808) for transmission of PAUSE frames. The upper 16 bits of this register are not reset and you must initialize it.



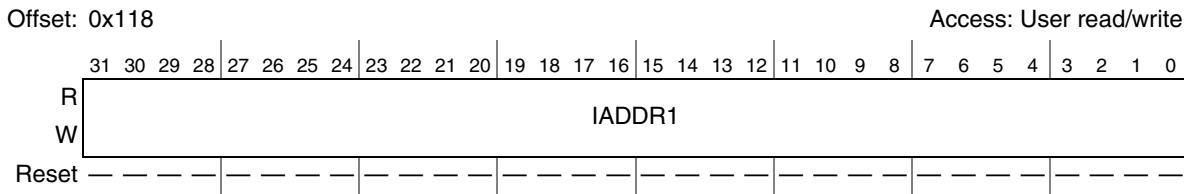


Figure 16-14. Descriptor Individual Upper Address Register (IAUR)

Table 16-16. IAUR Field Descriptions

Field	Description
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.

#### 16.4.14 Descriptor Individual Lower Address Register (IALR)

IALR contains the lower 32 bits of the 64-bit individual address hash table. The address recognition process uses this table to check for a possible match with the DA field of receive frames with an individual DA. This register is not reset and you must initialize it.

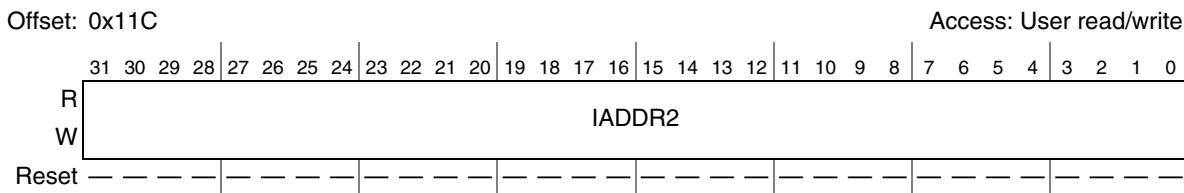


Figure 16-15. Descriptor Individual Lower Address Register (IALR)

Table 16-17. IALR Field Descriptions

Field	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.

#### 16.4.15 Descriptor Group Upper Address Register (GAUR)

GAUR contains the upper 32 bits of the 64-bit hash table used in the address recognition process for receive frames with a multicast address. You must initialize this register.

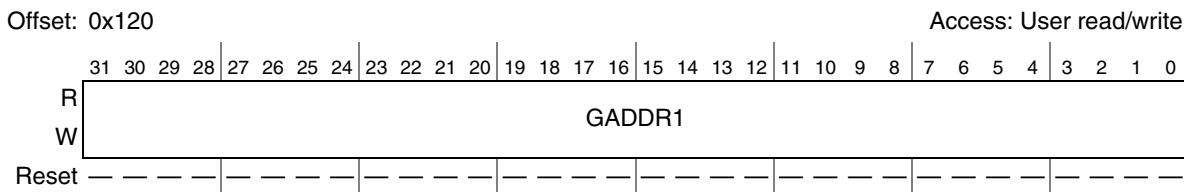


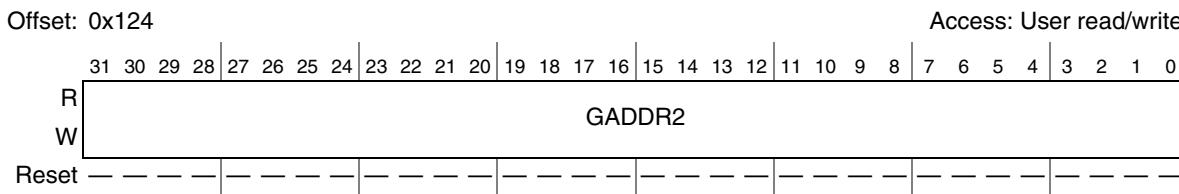
Figure 16-16. Descriptor Group Upper Address Register (GAUR)

**Table 16-18. GAUR Field Descriptions**

Field	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.

#### 16.4.16 Descriptor Group Lower Address Register (GALR)

GALR contains the lower 32 bits of the 64-bit hash table used in the address recognition process for receive frames with a multicast address. You must initialize this register.



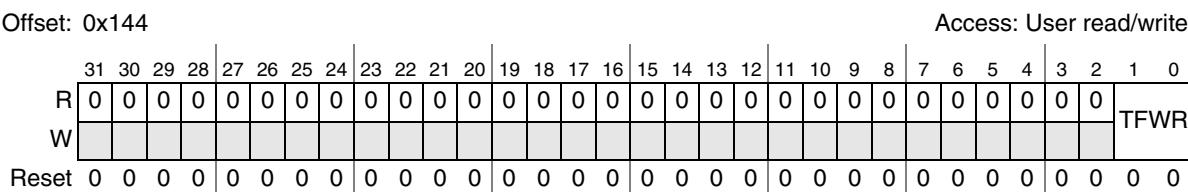
**Figure 16-17. Descriptor Group Lower Address Register (GALR)**

**Table 16-19. GALR Field Descriptions**

Field	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.

#### 16.4.17 Transmit FIFO Watermark Register (TFWR)

The TFWR controls the amount of data required in the transmit FIFO before transmission of a frame can begin. This allows you to minimize transmit latency (TFWR = 00 or 01) or allow for larger bus access latency (TFWR = 11) due to contention for the system bus. Setting the watermark to a high value minimizes the risk of transmit FIFO underrun due to contention for the system bus. The byte counts associated with the TFWR field may need to be modified to match a given system requirement (worst case bus access latency by the transmit data DMA channel).



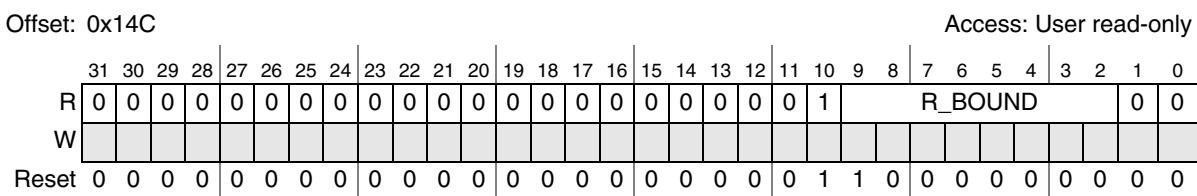
**Figure 16-18. Transmit FIFO Watermark Register (TFWR)**

**Table 16-20. TFWR Field Descriptions**

Field	Description
31–2	Reserved, must be cleared.
1–0 TFWR	Number of bytes written to transmit FIFO before transmission of a frame begins 00 64 bytes written 01 64 bytes written 10 128 bytes written 11 192 bytes written

#### **16.4.18 FIFO Receive Bound Register (FRBR)**

FRBR indicates the upper address bound of the FIFO RAM. Drivers can use this value, along with the FRSR, to appropriately divide the available FIFO RAM between the transmit and receive data paths.



**Figure 16-19. FIFO Receive Bound Register (FRBR)**

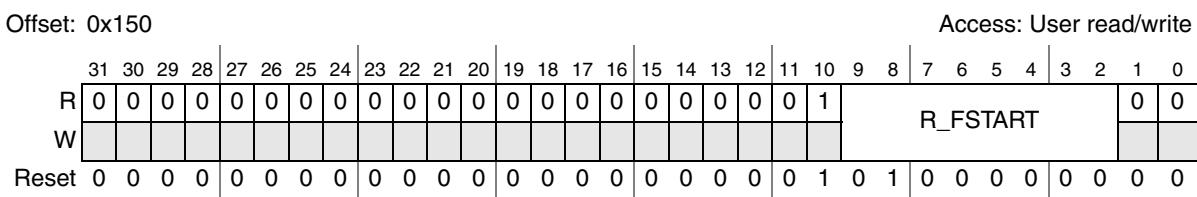
## Table 16-21. FRBR Field Descriptions

Field	Description
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, read as 0.

#### **16.4.19 FIFO Receive Start Register (FRSR)**

FRSR indicates the starting address of the receive FIFO. FRSR marks the boundary between the transmit and receive FIFOs. The transmit FIFO uses addresses from the start of the FIFO to the location four bytes before the address programmed into the FRSR. The receive FIFO uses addresses from FRSR to FRBR inclusive.

Hardware initializes the FRSR register at reset. FRSR only needs to be written to change the default value.



**Figure 16-20. FIFO Receive Start Register (FRSR)**

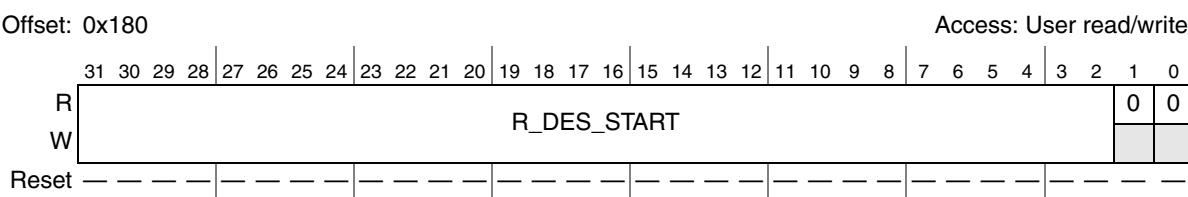
**Table 16-22. FRSR Field Descriptions**

Field	Description
31–11	Reserved, must be cleared.
10	Reserved, must be set.
9–2 R_FSTART	Address of first receive FIFO location. Acts as delimiter between receive and transmit FIFOs. For proper operation, ensure that R_FSTART is set to 0x48 or greater.
1–0	Reserved, must be cleared.

#### **16.4.20 Receive Descriptor Ring Start Register (ERDSR)**

ERDSR points 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 prior to operation.



**Figure 16-21. Ethernet Receive Descriptor Ring Start Register (ERDSR)**

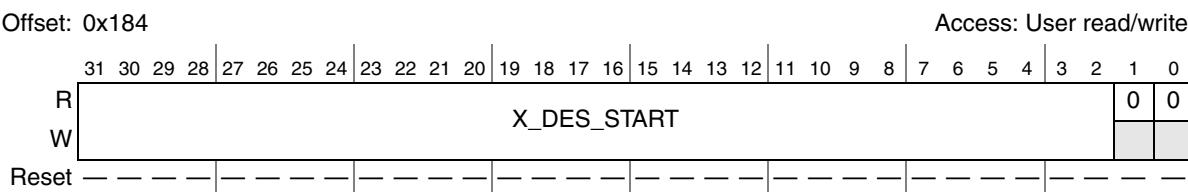
**Table 16-23. ERDSR Field Descriptions**

Field	Description
31–2 R_DES_START	Pointer to start of receive buffer descriptor queue.
1–0	Reserved, must be cleared.

#### 16.4.21 Transmit Buffer Descriptor Ring Start Registers (ETSDR)

ETSDR 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). You should write zeros to bits 1 and 0. Hardware ignores non-zero values in these two bit positions.

This register is undefined at reset and must be initialized prior to operation.



**Figure 16-22. Transmit Buffer Descriptor Ring Start Register (ETDSR)**

**Table 16-24. ETDSR Field Descriptions**

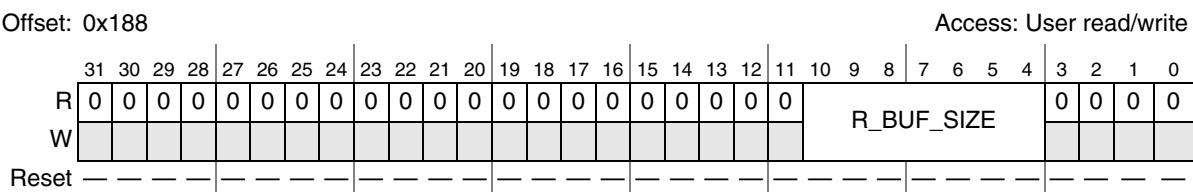
Field	Description
31–2 X_DES_START	Pointer to start of transmit buffer descriptor queue.
1–0	Reserved, must be cleared.

#### 16.4.22 Receive Buffer Size Register (EMRBR)

The EMRBR is a user-programmable register that dictates the maximum size of all receive buffers. This value should take into consideration that the receive CRC is always written into the last receive buffer. To allow one maximum size frame per buffer, EMRBR must be set to RCR[MAX\_FL] or larger. To properly align the buffer, EMRBR must be evenly divisible by 16. To ensure this, bits 3–0 are forced low.

To minimize bus utilization (descriptor fetches), it is recommended that EMRBR be greater than or equal to 256 bytes.

The EMRBR register is undefined at reset and must be initialized by the user.



**Figure 16-23. Receive Buffer Size Register (EMRBR)**

**Table 16-25. EMRBR Field Descriptions**

Field	Description
31–11	Reserved, must be cleared.
10–4 R_BUF_SIZE	Maximum size of receive buffer size in bytes. To minimize bus utilization (descriptor fetches), set this field to 256 bytes (0x10) or larger. 0x10 256 + 15 bytes (minimum size recommended) 0x11 272 + 15 bytes ... 0x7F 2032 + 15 bytes. The FEC writes up to 2047 bytes in the receive buffer. If data larger than 2047 is received, the FEC truncates it and shows 0x7FF in the receive descriptor
3–0	Reserved, must be cleared.

## 16.5 Functional Description

This section describes the operation of the FEC, beginning with the buffer descriptors, the hardware and software initialization sequence, then the software (Ethernet driver) interface for transmitting and receiving frames.

Following the software initialization and operation sections are sections providing a detailed description of the functions of the FEC.

## 16.5.1 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.

### 16.5.1.1 Driver/DMA Operation with Buffer Descriptors

The data for the FEC frames resides in one or more memory buffers external to the FEC. 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 by the FEC DMA engine.

Software produces buffers by allocating/initializing memory and initializing buffer descriptors. Setting the RxBD[E] or TxBD[R] bit produces the buffer. Software writing to TDAR or RDAR tells the FEC that a buffer is 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 DMA engine writes the buffer descriptor status bits, hardware clears RxBD[E] or TxBD[R] to signal the buffer has been consumed. Software may poll the BDs to detect when the buffers are consumed or may rely on the buffer/frame interrupts. The driver may process these buffers, and they can return to the free list.

The ECR[ETHER\_EN] bit operates as a reset to the BD/DMA logic. When ECR[ETHER\_EN] is cleared, 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 ECR[ETHER\_EN] is set.

The buffer descriptors operate as two separate rings. ERDSR defines the starting address for receive BDs and ETDSR defines the starting address for transmit BDs. The wrap (W) bit defines the last buffer descriptor in each ring. When W is set, the next descriptor in the ring is at the location pointed to by ERDSR and ETDSR for the receive and transmit rings, respectively. Buffer descriptor rings must start on a 32-bit boundary; however, it is recommended they are made 128-bit aligned.

#### 16.5.1.1.1 Driver/DMA Operation with Transmit BDs

Typically, a transmit frame is divided between multiple buffers. An example is to have an application payload in one buffer, TCP header in a second buffer, IP header in a third buffer, and Ethernet/IEEE 802.3 header in a fourth buffer. The Ethernet MAC does not prepend the Ethernet header (destination address, source address, length/type field(s)), so the driver must provide this in one of the transmit buffers. The Ethernet MAC can append the Ethernet CRC to the frame. TxBD[TC], which must be set by the driver, determines whether the MAC or driver appends the CRC.

The driver (TxBD software producer) should set up Tx BDs so a complete transmit frame is given to the hardware at once. If a transmit frame consists of three buffers, the BDs should be initialized with pointer, length, and control (W, L, TC, ABC) and then the TxBD[R] bit should be set in reverse order (third, second, then first BD) to ensure that the complete frame is ready in memory before the DMA begins. If the TxBDs are set up in order, the DMA controller could DMA the first BD before the second was made available, potentially causing a transmit FIFO underrun.

In the FEC, the driver notifies the DMA that new transmit frame(s) are available by writing to TDAR. When this register is written to (data value is not significant) the FEC, RISC tells the DMA to read the next transmit BD in the ring. After started, the RISC + DMA continues to read and interpret transmit BDs in order and DMA the associated buffers until a transmit BD is encountered with the R bit cleared. At this point, the FEC polls this BD one more time. If the R bit is cleared the second time, RISC stops the transmit descriptor read process until software sets up another transmit frame and writes to TDAR.

When the DMA of each transmit buffer is complete, the DMA writes back to the BD to clear the R bit, indicating that the hardware consumer is finished with the buffer.

#### **16.5.1.2 Driver/DMA Operation with Receive BDs**

Unlike transmit, the length of the receive frame is unknown by the driver ahead of time. Therefore, the driver must set a variable to define the length of all receive buffers. In the FEC, this variable is written to the EMRBR register.

The driver (RxBD software producer) should set up some number of empty buffers for the Ethernet by initializing the address field and the E and W bits of the associated receive BDs. The hardware (receive DMA) consumes these buffers by filling them with data as frames are received and clearing the E bit and writing to the L bit (1 indicates last buffer in frame), the frame status bits (if L is set), and the length field.

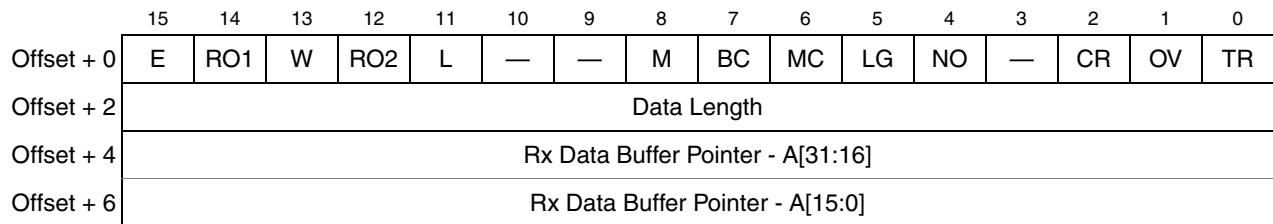
If a receive frame spans multiple receive buffers, the L bit is only set for the last buffer in the frame. For non-last buffers, the length field in the receive BD is written by the DMA (at the same time the E bit is cleared) with the default receive buffer length value. For end-of-frame buffers, the receive BD is written with L set and information written to the status bits (M, BC, MC, LG, NO, CR, OV, TR). Some of the status bits are error indicators which, if set, indicate the receive frame should be discarded and not given to higher layers. The frame status/length information is written into the receive FIFO following the end of the frame (as a single 32-bit word) by the receive logic. The length field for the end of frame buffer is written with the length of the entire frame, not only the length of the last buffer.

For simplicity, the driver may assign a large enough default receive buffer length 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 2047 bytes so software never sees a receive frame larger than 2047 bytes.

Similar to transmit, the FEC polls the receive descriptor ring after the driver sets up receive BDs and writes to the RDAR register. As frames are received, the FEC fills receive buffers and updates the associated BDs, then reads the next BD in the receive descriptor ring. If the FEC reads a receive BD and finds the E bit cleared, it polls this BD once more. If RxBD[E] is clear a second time, FEC stops reading receive BDs until the driver writes to RDAR.

#### **16.5.1.2 Ethernet Receive Buffer Descriptor (RxBD)**

In the RxBD, the user initializes the E and W bits in the first longword and the pointer in the second longword. When the buffer has been DMA'd, the Ethernet controller modifies the E, L, M, BC, MC, LG, NO, CR, OV, and TR bits and writes the length of the used portion of the buffer in the first 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.

**Figure 16-24. Receive Buffer Descriptor (RxBD)****Table 16-26. Receive Buffer Descriptor Field Definitions**

Word	Field	Description
Offset + 0	15 E	Empty. Written by the FEC (=0) and user (=1). 0 The data buffer associated with this BD is filled with received data, or data reception has 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 is not modified by hardware, nor does 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.
Offset + 0	12 RO2	Receive software ownership. This field is reserved for use by software. This read/write bit is not modified by hardware, nor does 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, must be cleared.
Offset + 0	8 M	Miss. Written by the FEC. This bit is set by the FEC for frames accepted in promiscuous mode, but flagged as a miss by the internal address recognition. Therefore, while in promiscuous mode, you 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	Set if the DA is broadcast (FFFF_FFFF_FFFF).
Offset + 0	6 MC	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 RCR[MAX_FL] was recognized. This bit is valid only if the L-bit is set. The receive data is not altered in any way unless the length exceeds 2047 bytes.
Offset + 0	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 is not set.
Offset + 0	3	Reserved, must be cleared.
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.

**Table 16-26. Receive Buffer Descriptor Field Definitions (continued)**

Word	Field	Description
Offset + 0	1 OV	Overrun. Written by the FEC. A receive FIFO overrun occurred during frame reception. If this bit is set, the other status bits, M, LG, NO, CR, and CL lose their normal meaning and are zero. This bit is valid only if the L-bit is set.
Offset + 0	0 TR	Set if the receive frame is truncated (frame length > 2047 bytes). If the TR bit is set, the frame must be discarded and the other error bits must 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 equals 0 (the value is equal to EMRBR), or the length of the frame including CRC if L is set. 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, containing the address of the associated data buffer, must always be evenly divisible by 16. The buffer must reside in memory external to the FEC. The Ethernet controller never modifies this value.

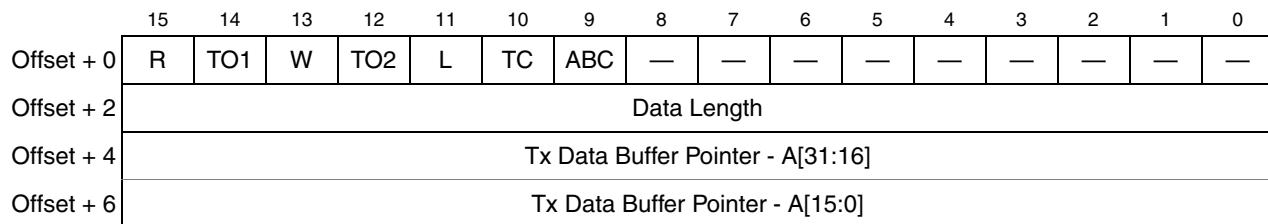
### NOTE

When the software driver sets an E bit in one or more receive descriptors, the driver should follow with a write to RDAR.

#### 16.5.1.3 Ethernet Transmit Buffer Descriptor (TxBD)

Data is presented to the FEC for transmission by arranging it in buffers referenced by the channel's TxBDs. The Ethernet controller confirms transmission by clearing the ready bit (TxBD[R]) when DMA of the buffer is complete. In the TxBD, the user initializes the R, W, L, and TC bits and the length (in bytes) in the first longword and the buffer pointer in the second longword.

The FEC clears the R bit when the buffer is transferred. 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).

**Figure 16-25. Transmit Buffer Descriptor (TxBD)**

**Table 16-27. Transmit Buffer Descriptor Field Definitions**

<b>Word</b>	<b>Field</b>	<b>Description</b>
Offset + 0	15 R	Ready. Written by the FEC and you. 0 The data buffer associated with this BD is not ready for transmission. You are 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, prepared for transmission by you, has not been transmitted or currently transmits. You may write no fields of this BD after this bit is set.
Offset + 0	14 TO1	Transmit software ownership. This field is reserved for software use. This read/write bit is not modified by hardware nor does 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.
Offset + 0	12 TO2	Transmit software ownership. This field is reserved for use by software. This read/write bit is not modified by hardware nor does 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	Transmit CRC. Written by user (only valid if L is set). 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 is set). 0 No effect 1 Transmit the CRC sequence inverted after the last data byte (regardless of TC value)
Offset + 0	8–0	Reserved, must be cleared.
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.
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, containing 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

After 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 is setting the R bit in the first BD for the frame. The driver must follow that with a write to TDAR that triggers the FEC to poll the next BD in the ring.

## 16.5.2 Initialization Sequence

This section describes which registers are reset due to hardware reset, which are reset by the FEC RISC, and what locations you must initialize prior to enabling the FEC.

### 16.5.2.1 Hardware Controlled Initialization

In the FEC, hardware resets registers and control logic that generate interrupts. A hardware reset negates output signals and resets general configuration bits.

Other registers reset when the ECR[ETHER\_EN] bit is cleared (which is accomplished by a hard reset or software to halt operation). By clearing ECR[ETHER\_EN], configuration control registers such as the TCR and RCR are not reset, but the entire data path is reset.

**Table 16-28. ECR[ETHER\_EN] De-Assertion Effect on FEC**

Register/Machine	Reset Value
XMIT block	Transmission is aborted (bad CRC appended)
RECV block	Receive activity is aborted
DMA block	All DMA activity is terminated
RDAR	Cleared
TDAR	Cleared
Descriptor Controller block	Halt operation

### 16.5.3 User Initialization (Prior to Setting ECR[ETHER\_EN])

You need to initialize portions of the FEC prior to setting the ECR[ETHER\_EN] bit. The exact values depend on the particular application. The sequence is not important.

Table 16-29 defines Ethernet MAC registers requiring initialization.

**Table 16-29. User Initialization (Before ECR[ETHER\_EN])**

Description
Initialize EIMR
Clear EIR (write 0xFFFF_FFFF)
TFWR (optional)
IALR / IAUR
GAUR / GALR
PALR / PAUR (only needed for full duplex flow control)
OPD (only needed for full duplex flow control)
RCR
TCR
MSCR (optional)

Table 16-30 defines FEC FIFO/DMA registers that require initialization.

**Table 16-30. FEC User Initialization (Before ECR[ETHER\_EN])**

<b>Description</b>
Initialize FRSR (optional)
Initialize EMRBR
Initialize ERDSR
Initialize ETDSR
Initialize (Empty) Transmit Descriptor ring
Initialize (Empty) Receive Descriptor ring

### 16.5.4 Microcontroller Initialization

In the FEC, the descriptor control RISC initializes some registers after ECR[ETHER\_EN] is asserted. After the microcontroller initialization sequence is complete, hardware is ready for operation.

[Table 16-31](#) shows microcontroller initialization operations.

**Table 16-31. Microcontroller Initialization**

<b>Description</b>
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

### 16.5.5 User Initialization (After Setting ECR[ETHER\_EN])

After setting ECR[ETHER\_EN], you can set up the buffer/frame descriptors and write to TDAR and RDAR. Refer to [Section 16.5.1, “Buffer Descriptors,”](#) for more details.

### 16.5.6 Network Interface Options

The FEC supports an MII interface for 10/100 Mbps Ethernet and a 7-wire serial interface for 10 Mbps Ethernet. The RCR[MII\_MODE] bit select the interface mode. In MII mode (RCR[MII\_MODE] set), there are 18 signals defined by the IEEE 802.3 standard and supported by the EMAC. [Table 16-32](#) shows these signals.

**Table 16-32. MII Mode**

Signal Description	EMAC pin
Transmit Clock	FEC_TXCLK
Transmit Enable	FEC_TXEN
Transmit Data	FEC_TXD[3:0]
Transmit Error	FEC_RXER
Collision	FEC_COL
Carrier Sense	FEC_CRS
Receive Clock	FEC_RXCLK
Receive Data Valid	FEC_RXDV
Receive Data	FEC_RXD[3:0]
Receive Error	FEC_RXER
Management Data Clock	FEC_MDC
Management Data Input/Output	FEC_MDIO

The 7-wire serial mode interface (RCR[MII\_MODE] cleared) is generally referred to as AMD mode. [Table 16-33](#) shows the 7-wire mode connections to the external transceiver.

**Table 16-33. 7-Wire Mode Configuration**

Signal description	EMAC Pin
Transmit Clock	FEC_TXCLK
Transmit Enable	FEC_TXEN
Transmit Data	FEC_TXD[0]
Collision	FEC_COL
Receive Clock	FEC_RXCLK
Receive Data Valid	FEC_RXDV
Receive Data	FEC_RXD[0]

## 16.5.7 FEC Frame Transmission

The Ethernet transmitter is designed to work with almost no intervention from software. After ECR[ETHER\_EN] is set and data appears in the transmit FIFO, the Ethernet MAC can transmit onto the network. The Ethernet controller transmits bytes least significant bit (lsb) first.

When the transmit FIFO fills to the watermark (defined by TFWR), MAC transmit logic asserts FEC\_TXEN and starts transmitting the preamble (PA) sequence, the start frame delimiter (SFD), and then the frame information from the FIFO. However, the controller defers the transmission if the network is

busy (FEC\_CRS is asserted). Before transmitting, the controller waits for carrier sense to become inactive, then determines if carrier sense stays inactive for 60 bit times. If so, transmission begins after waiting an additional 36 bit times (96 bit times after carrier sense originally became inactive). See [Section 16.5.15.1, “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 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 is transmitted, FCS (frame check sequence) or 32-bit cyclic redundancy check (CRC) bytes are appended if the TC bit is set in the transmit frame control word. If the ABC bit is set in the transmit frame control word, a bad CRC is appended to the frame data regardless of the TC bit value. Transmit logic automatically pads short frames (if the TC bit in the transmit buffer descriptor for the end of frame buffer is set).

Settings in the EIMR determine interrupts generated to the buffer (TXB) and frame (TFINT).

The transmit error interrupts are HBERR, BABT, LATE\_COL, COL\_RETRY\_LIM, and XFIFO\_UN. If the transmit frame length exceeds MAX\_FL bytes, BABT interrupt is asserted. However, the entire frame is transmitted (no truncation).

To pause transmission, set TCR[GTS] (graceful transmit stop). The FEC transmitter stops immediately if transmission is not in progress; otherwise, it continues transmission until the current frame finishes or terminates with a collision. After the transmitter has stopped, the GRA (graceful stop complete) interrupt is asserted. If TCR[GTS] is cleared, the FEC resumes transmission with the next frame.

### 16.5.7.1 Duplicate Frame Transmission

The FEC fetches transmit buffer descriptors (TxBDs) and the corresponding transmit data continuously until the transmit FIFO is full. It does not determine whether the TxBD to be fetched is already being processed internally (as a result of a wrap). As the FEC nears the end of the transmission of one frame, it begins to DMA the data for the next frame. To remain one BD ahead of the DMA, it also fetches the TxBD for the next frame. It is possible that the FEC fetches from memory a BD that has already been processed but not yet written back (it is read a second time with the R bit remains set). In this case, the data is fetched and transmitted again.

Using at least three TxBDs fixes this problem for large frames, but not for small frames. To ensure correct operation for large or small frames, one of the following must be true:

- The FEC software driver ensures that there is always at least one TxBD with the ready bit cleared.
- Every frame uses more than one TxBD and every TxBD but the last is written back immediately after the data is fetched.
- The FEC software driver ensures a minimum frame size,  $n$ . The minimum number of TxBDs is then  $(\text{Tx FIFO Size} \div (n + 4))$  rounded up to the nearest integer (though the result cannot be less than three). The default Tx FIFO size is 192 bytes; this size is programmable.

## 16.5.8 FEC Frame Reception

The FEC receiver works with almost no intervention from the host and can perform address recognition, CRC checking, short frame checking, and maximum frame length checking. The Ethernet controller receives serial data lsb first.

When the driver enables the FEC receiver by setting ECR[ETHER\_EN], it immediately starts processing receive frames. When FEC\_RXDV is asserted, the receiver first checks for a valid PA/SFD header. If the PA/SFD is valid, it is stripped and the receiver processes the frame. If a valid PA/SFD is not found, the frame is ignored.

In serial mode, the first 16 bit times of RX\_D0 following assertion of FEC\_RXDV 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 are received, the FEC performs address recognition on the frame.

After a collision window (64 bytes) of data is received and if address recognition has not rejected the frame, the receive FIFO signals 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. Therefore, no collision fragments are presented to you except late collisions, which indicate serious LAN problems.

During reception, the Ethernet controller checks for various error conditions and after 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 16.5.15.2, “Reception Errors,”](#) for more details.

Receive buffer (RXB) and frame interrupts (RFINT) may be generated if enabled by the EIMR register. A receive error interrupt is a babbling receiver error (BABR). Receive frames are not truncated if they exceed the max frame length (MAX\_FL); however, the BABR interrupt occurs and the LG bit in the receive buffer descriptor (RxBD) is set. See [Section 16.5.1.2, “Ethernet Receive Buffer Descriptor \(RxBD\),”](#) for more details.

When the receive frame is complete, the FEC sets the L-bit in the RxBD, writes the other frame status bits into the RxBD, and clears the E-bit. The Ethernet controller next generates a maskable interrupt (RFINT bit in EIR, maskable by RFIEN bit in EIMR), indicating that a frame is received and is in memory. The Ethernet controller then waits for a new frame.

## 16.5.9 Ethernet Address Recognition

The FEC filters 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 appears 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 16-26](#) illustrates the address recognition decisions made by the receive block, while [Figure 16-27](#) illustrates the decisions made by the microcontroller.

If the DA is a broadcast address and broadcast reject (RCR[BC\_REJ]) is cleared, then the frame is accepted unconditionally, as shown in [Figure 16-26](#). Otherwise, if the DA is not a broadcast address, then the microcontroller runs the address recognition subroutine, as shown in [Figure 16-27](#).

If the DA is a group (multicast) address and flow control is disabled, then the microcontroller performs a group hash table lookup using the 64-entry hash table programmed in GAUR and GALR. If a hash match occurs, the receiver accepts the frame.

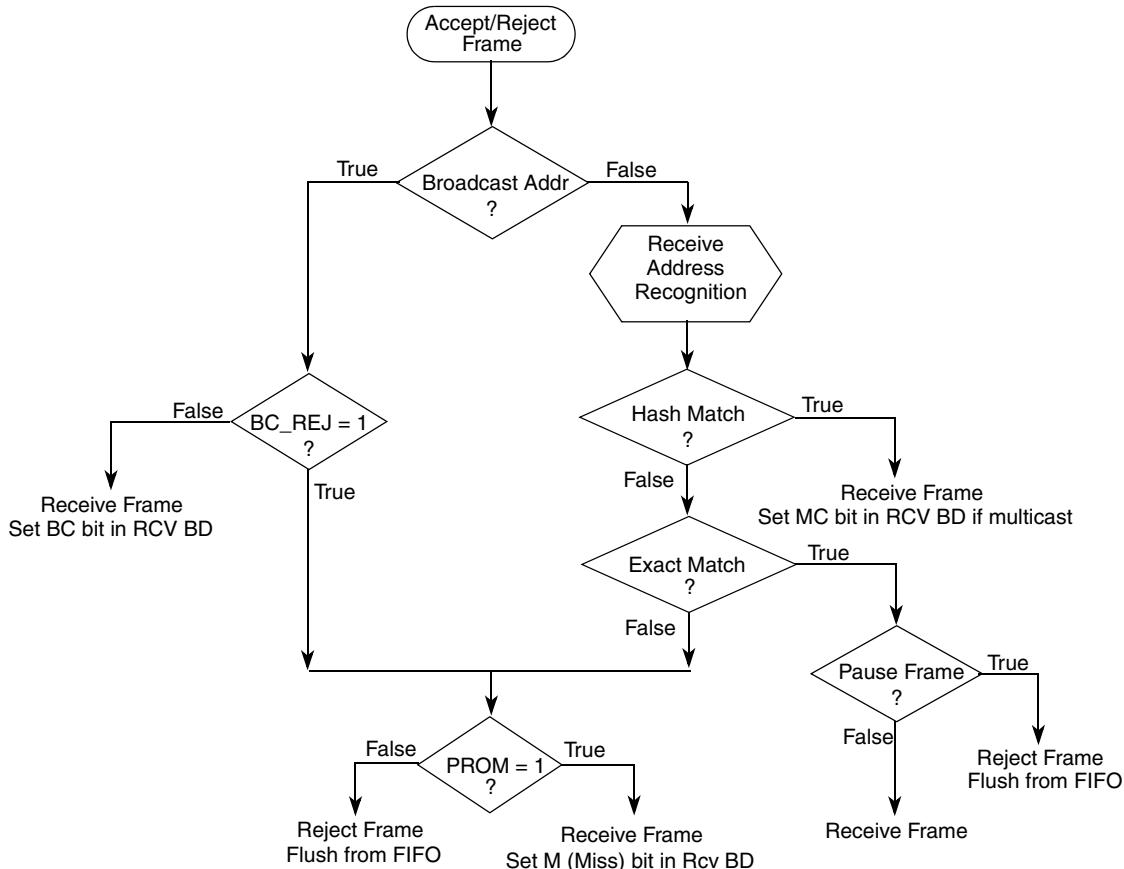
If flow control is enabled, the microcontroller does an exact address match check between the DA and the designated PAUSE DA (01:80:C2:00:00:01). If the receive block determines the received frame is a valid PAUSE frame, the frame is rejected. The receiver detects a PAUSE frame with the DA field set to 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 you program in the PALR and PAUR registers. If an exact match occurs, the frame is accepted; otherwise, the microcontroller does an individual hash table lookup using the 64-entry hash table programmed in registers, IAUR and IALR. In the case of an individual hash match, the frame is accepted. Again, the receiver accepts or rejects the frame based on PAUSE frame detection, shown in [Figure 16-26](#).

If neither a hash match (group or individual) nor an exact match (group or individual) occur, and if promiscuous mode is enabled (RCR[PROM] set), the frame is accepted and the MISS bit in the receive buffer descriptor is set; otherwise, the frame is rejected.

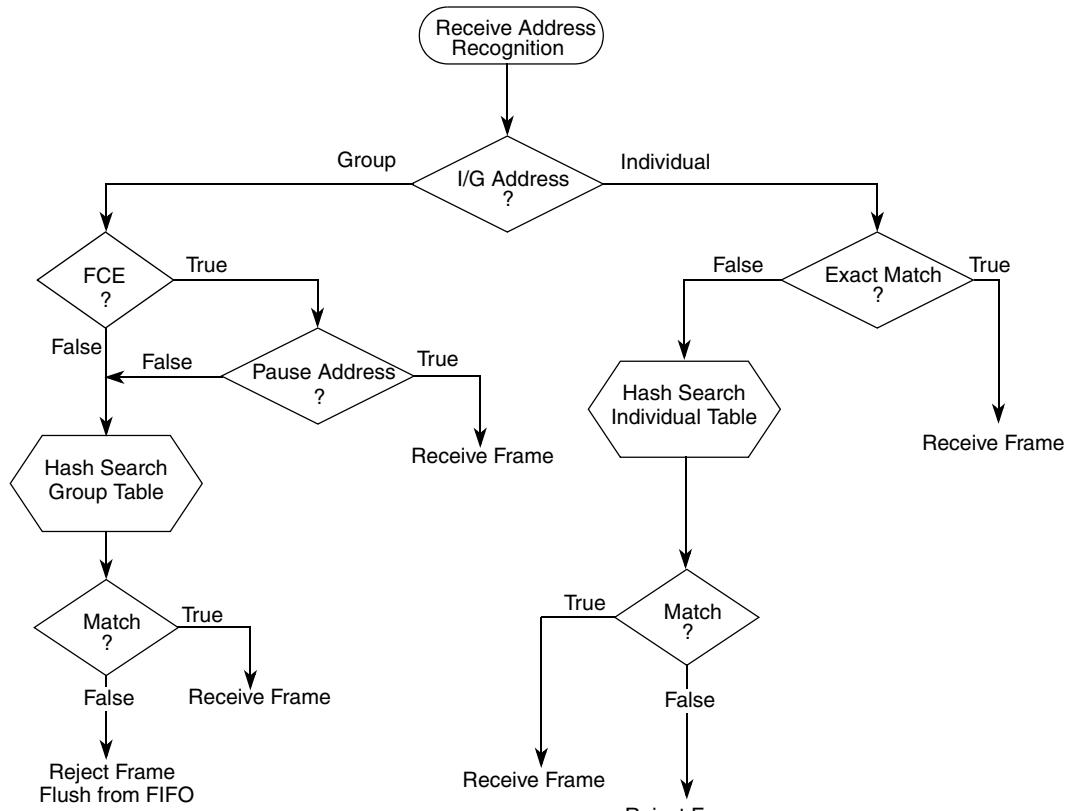
Similarly, if the DA is a broadcast address, broadcast reject (RCR[BC\_REJ]) is asserted, and promiscuous mode is enabled, the frame is accepted and the MISS bit in the receive buffer descriptor is set; otherwise, the frame is rejected.

In general, when a frame is rejected, it is flushed from the FIFO.

**Notes:**

BC\_REJ - field in RCR register (BroadCast REject)  
 PROM - field in RCR register (PROMiscous mode)  
 Pause Frame - valid PAUSE frame received

**Figure 16-26. Ethernet Address Recognition—Receive Block Decisions**

**Notes:**

FCE - field in RCR register (flow control enable)

I/G - Individual/Group bit in destination address (lsb in first byte received in MAC frame)

**Figure 16-27. Ethernet Address Recognition—Microcode Decisions**

### 16.5.10 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, represented by 64 bits stored in GAUR, GALR (group address hash match), or IAUR, IALR (individual address hash match). This mapping is performed by passing the 48-bit address through the on-chip 32-bit CRC generator and selecting the six most significant bits of the CRC-encoded result to generate a number between 0 and 63. The msb of the CRC result selects GAUR (msb = 1) or GALR (msb = 0). The five least significant bits of the hash result select the bit within the selected register. If the CRC generator selects a bit 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 (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 user must initialize the hash table registers. Use this CRC32 polynomial to compute the hash:

$$X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^8 + X^7 + X^5 + X^4 + X^2 + X + 1$$

**Eqn. 16-2**

Table 16-34 contains example destination addresses and corresponding hash values.

**Table 16-34. Destination Address to 6-Bit Hash**

48-bit DA	6-bit Hash (in hex)	Hash Decimal Value
65FF_FFFF_FFFF	0x0	0
55FF_FFFF_FFFF	0x1	1
15FF_FFFF_FFFF	0x2	2
35FF_FFFF_FFFF	0x3	3
B5FF_FFFF_FFFF	0x4	4
95FF_FFFF_FFFF	0x5	5
D5FF_FFFF_FFFF	0x6	6
F5FF_FFFF_FFFF	0x7	7
DBFF_FFFF_FFFF	0x8	8
FBFF_FFFF_FFFF	0x9	9
BBFF_FFFF_FFFF	0xA	10
8BFF_FFFF_FFFF	0xB	11
0BFF_FFFF_FFFF	0xC	12
3BFF_FFFF_FFFF	0xD	13
7BFF_FFFF_FFFF	0xE	14
5BFF_FFFF_FFFF	0xF	15
27FF_FFFF_FFFF	0x10	16
07FF_FFFF_FFFF	0x11	17
57FF_FFFF_FFFF	0x12	18
77FF_FFFF_FFFF	0x13	19
F7FF_FFFF_FFFF	0x14	20
C7FF_FFFF_FFFF	0x15	21
97FF_FFFF_FFFF	0x16	22
A7FF_FFFF_FFFF	0x17	23
99FF_FFFF_FFFF	0x18	24
B9FF_FFFF_FFFF	0x19	25
F9FF_FFFF_FFFF	0x1A	26
C9FF_FFFF_FFFF	0x1B	27

**Table 16-34. Destination Address to 6-Bit Hash (continued)**

<b>48-bit DA</b>	<b>6-bit Hash (in hex)</b>	<b>Hash Decimal Value</b>
59FF_FFFF_FFFF	0x1C	28
79FF_FFFF_FFFF	0x1D	29
29FF_FFFF_FFFF	0x1E	30
19FF_FFFF_FFFF	0x1F	31
D1FF_FFFF_FFFF	0x20	32
F1FF_FFFF_FFFF	0x21	33
B1FF_FFFF_FFFF	0x22	34
91FF_FFFF_FFFF	0x23	35
11FF_FFFF_FFFF	0x24	36
31FF_FFFF_FFFF	0x25	37
71FF_FFFF_FFFF	0x26	38
51FF_FFFF_FFFF	0x27	39
7FFF_FFFF_FFFF	0x28	40
4FFF_FFFF_FFFF	0x29	41
1FFF_FFFF_FFFF	0x2A	42
3FFF_FFFF_FFFF	0x2B	43
BFFF_FFFF_FFFF	0x2C	44
9FFF_FFFF_FFFF	0x2D	45
DFFF_FFFF_FFFF	0x2E	46
EFFF_FFFF_FFFF	0x2F	47
93FF_FFFF_FFFF	0x30	48
B3FF_FFFF_FFFF	0x31	49
F3FF_FFFF_FFFF	0x32	50
D3FF_FFFF_FFFF	0x33	51
53FF_FFFF_FFFF	0x34	52
73FF_FFFF_FFFF	0x35	53
23FF_FFFF_FFFF	0x36	54
13FF_FFFF_FFFF	0x37	55
3DFF_FFFF_FFFF	0x38	56
0DFF_FFFF_FFFF	0x39	57
5DFF_FFFF_FFFF	0x3A	58
7DFF_FFFF_FFFF	0x3B	59

**Table 16-34. Destination Address to 6-Bit Hash (continued)**

<b>48-bit DA</b>	<b>6-bit Hash (in hex)</b>	<b>Hash Decimal Value</b>
FDFF_FFFF_FFFF	0x3C	60
DDFF_FFFF_FFFF	0x3D	61
9DFF_FFFF_FFFF	0x3E	62
BDFF_FFFF_FFFF	0x3F	63

### 16.5.11 Full Duplex Flow Control

Full-duplex flow control allows you to transmit pause frames and to detect received pause frames. Upon detection of a pause frame, MAC data frame transmission stops for a given pause duration.

To enable PAUSE frame detection, the FEC must operate in full-duplex mode (TCR[FDEN] set) with flow control (RCR[FCE] set). The FEC detects a pause frame when the fields of the incoming frame match the pause frame specifications, as shown in [Table 16-35](#). In addition, the receive status associated with the frame should indicate that the frame is valid.

**Table 16-35. PAUSE Frame Field Specification**

<b>48-bit Destination Address</b>	0x0180_C200_0001 or Physical Address
<b>48-bit Source Address</b>	Any
<b>16-bit Type</b>	0x8808
<b>16-bit Opcode</b>	0x0001
<b>16-bit PAUSE Duration</b>	0x0000 – 0xFFFF

The receiver and microcontroller modules perform PAUSE frame detection. The microcontroller runs an address recognition subroutine to detect the specified pause frame destination address, while the receiver detects the type and opcode pause frame fields. On detection of a pause frame, TCR[GTS] is set by the FEC internally. When transmission has paused, the EIR[GRA] interrupt is asserted and the pause timer begins to increment. The pause timer uses the transmit backoff timer hardware for tracking the appropriate collision backoff time in half-duplex mode. The pause timer increments once every slot time, until OPD[PAUSE\_DUR] slot times have expired. On OPD[PAUSE\_DUR] expiration, TCR[GTS] is cleared allowing MAC data frame transmission to resume. The receive flow control pause status bit (TCR[RFC\_PAUSE]) is set while the transmitter pauses due to reception of a pause frame.

To transmit a pause frame, the FEC must operate in full-duplex mode and you must set flow control pause (TCR[TFC\_PAUSE]). After TCR[TFC\_PAUSE] is set, the transmitter sets TCR[GTS] internally. When the transmission of data frames stops, the EIR[GRA] (graceful stop complete) interrupt asserts and the pause frame is transmitted. TCR[TFC\_PAUSE,GTS] are then cleared internally.

You must specify the desired pause duration in the OPD register.

When the transmitter pauses due to receiver/microcontroller pause frame detection, TCR[TFC\_PAUSE] may remain set and cause the transmission of a single pause frame. In this case, the EIR[GRA] interrupt is not asserted.

### 16.5.12 Inter-Packet Gap (IPG) Time

The minimum inter-packet gap time for back-to-back transmission is 96 bit times. After completing a transmission or after the backoff algorithm completes, the transmitter waits for carrier sense to be negated before starting its 96 bit time IPG counter. Frame transmission may begin 96 bit times after carrier sense is negated if it stays negated for at least 60 bit times. If carrier sense asserts during the last 36 bit times, it is ignored and a collision occurs.

The receiver accepts 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 receiver may discard the following frame.

### 16.5.13 Collision Managing

If a collision occurs during frame transmission, the Ethernet controller continues the transmission for at least 32 bit times, transmitting a JAM pattern consisting of 32 ones. If the collision occurs during the preamble sequence, a JAM pattern is sent after the end of the preamble sequence.

If a collision occurs within 512 bit times (one slot time), the retry process is initiated. The transmitter waits a random number of slot 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.

### 16.5.14 MII Internal and External Loopback

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. Internal and external loopback are configured using combinations of the RCR[LOOP, DRT] and TCR[FDEN] bits.

Set FDEN for internal and external loopback.

For internal loopback, set RCR[LOOP] and clear RCR[DRT]. FEC\_TXEN and FEC\_TXER do not assert during internal loopback. During internal loopback, the transmit/receive data rate is higher than in normal operation because the transmit and receive blocks use the internal bus clock instead of the clocks from the external transceiver. This causes an increase in the required system bus bandwidth for transmit and receive data being DMA'd to/from external memory. It may be necessary to pace the frames on the transmit side and/or limit the size of the frames to prevent transmit FIFO underruns and receive FIFO overflows.

For external loopback, clear RCR[LOOP] and RCR[DRT], and configure the external transceiver for loopback.

### 16.5.15 Ethernet Error-Managing Procedure

The Ethernet controller reports frame reception and transmission error conditions using the FEC RxBDs and the EIR register.

## 16.5.15.1 Transmission Errors

### 16.5.15.1.1 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, and EIR[UN] is set. The FEC then continues to the next transmit buffer descriptor and begin transmitting the next frame. The UN interrupt is asserted if enabled in the EIMR register.

### 16.5.15.1.2 Retransmission Attempts Limit Expired

When this error occurs, the FEC terminates transmission. All remaining buffers for that frame are flushed and closed, and EIR[RL] is set. The FEC then continues to the next transmit buffer descriptor and begins transmitting the next frame. The RL interrupt is asserted if enabled in the EIMR register.

### 16.5.15.1.3 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 EIR[LC] is set. The FEC then continues to the next transmit buffer descriptor and begin transmitting the next frame. The LC interrupt is asserted if enabled in the EIMR register.

### 16.5.15.1.4 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 four 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 continues to function properly. This is the heartbeat condition.

If TCR[HBC] is set and the heartbeat condition is not detected by the FEC after a frame transmission, a heartbeat error occurs. When this error occurs, the FEC closes the buffer, sets EIR[HB], and generates the HBERR interrupt if it is enabled.

## 16.5.15.2 Reception Errors

### 16.5.15.2.1 Overrun Error

If the receive block has data to put into the receive FIFO and the receive FIFO is full, FEC sets RxBD[OV]. All subsequent data in the frame is 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. The driver must discard this frame.

### 16.5.15.2.2 Non-Octet Error (Dribbling Bits)

The Ethernet controller manages 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, the frame non-octet aligned (NO) error is reported in the RxBD. If there is no CRC error, no error is reported.

### 16.5.15.2.3 CRC Error

When a CRC error occurs with no dribble bits, FEC closes the buffer and sets RxBD[CR]. CRC checking cannot be disabled, but the CRC error can be ignored if checking is not required.

### 16.5.15.2.4 Frame Length Violation

When the receive frame length exceeds MAX\_FL bytes the BABR interrupt is generated, and RxBD[LG] is set. The frame is not truncated unless the frame length exceeds 2047 bytes.

### 16.5.15.2.5 Truncation

When the receive frame length exceeds 2047 bytes, frame is truncated and RxBD[TR] is set.

# Chapter 17

## Inter-Integrated Circuit (IIC)

### 17.1 Introduction

The inter-integrated circuit (IIC) provides a method of communication between a number of device. 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 clock/20, with reduced bus loading. The maximum communication length and the number of devices that can be connected are limited by a maximum bus capacitance of 400 pF. Support System Management Bus Specification (SMBus), version2 and filter registers.

#### NOTE

- Use pin mux control registers from [Section 2.3, “Pin Mux Controls”](#) to assign IIC signals to the MCF51CN128 package pins.
- Most pin functions default to GPIO and must be software configured before using IIC.

### 17.1.1 Features

The IIC includes these distinctive features:

- Compatible with IIC bus standard
- Multi-master operation
- Software programmable for one of 64 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/detection
- Acknowledge bit generation/detection
- Bus busy detection
- General call recognition
- 10-bit address extension
- Support System Management Bus Specification (SMBus), version2
- Programmable glitch input filter

### 17.1.2 Modes of Operation

A brief description of the IIC in the various MCU modes is given here.

- **Run mode** — This is the basic mode of operation. To conserve power in this mode, disable the module.
- **Wait mode** — The module continues to operate while the MCU is in wait mode and can provide a wakeup interrupt.
- **Stop mode** — The IIC is inactive in stop3 mode for reduced power consumption. The STOP instruction does not affect IIC register states. Stop2 resets the register contents.

### 17.1.3 Block Diagram

Figure 17-1 is a block diagram of the IIC.

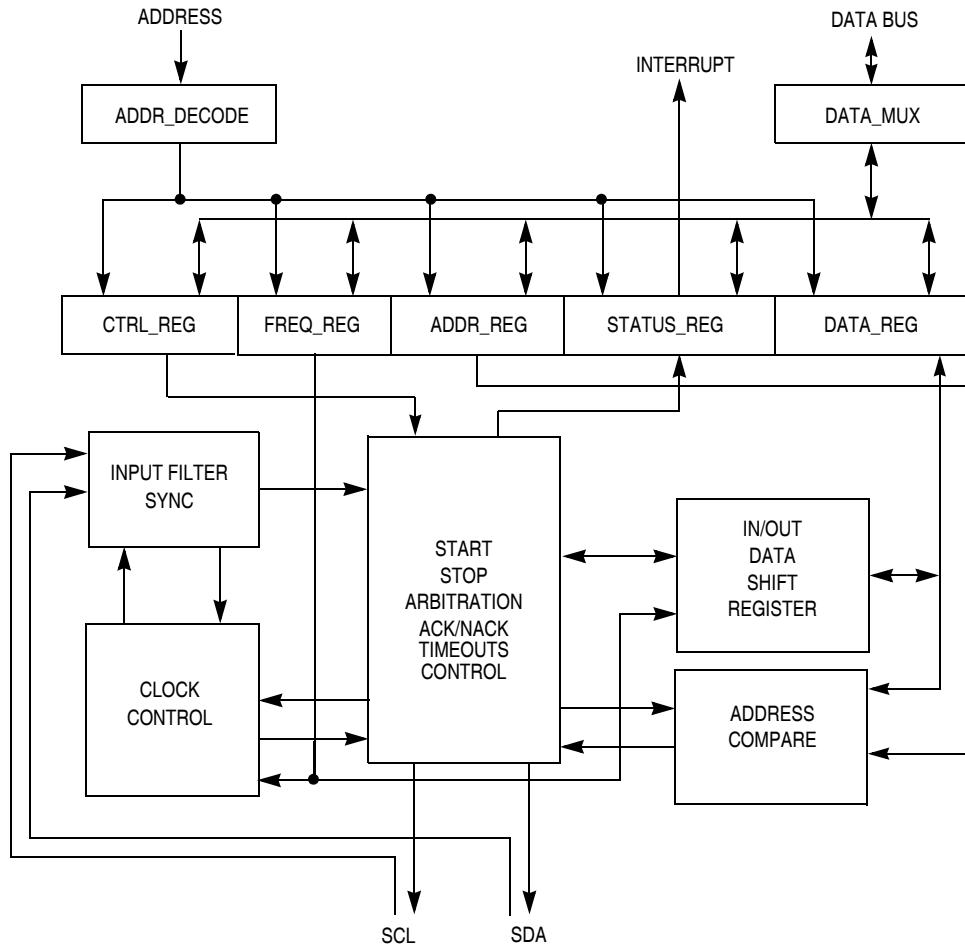


Figure 17-1. IIC Functional Block Diagram

## 17.2 External Signal Description

This section describes each user-accessible pin signal.

### 17.2.1 SCL — Serial Clock Line

The bidirectional SCL is the serial clock line of the IIC system.

### 17.2.2 SDA — Serial Data Line

The bidirectional SDA is the serial data line of the IIC system.

## 17.3 Register Definition

### 17.3.1 Module Memory Map

The IIC has 10 8-bit registers. The base address of the module is hardware programmable. The IIC register map is fixed and begins at the module's base address. [Table 17-1](#) summarizes the IIC module's address space. The following section describes the bit-level arrangement and functionality of each register.

**Table 17-1. Module Memory Map**

Address	Use	Access
Base + \$0000	IIC Address Register 1 (IICA1)	read/write
Base + \$0001	IIC Frequency Divider Register (IICF)	read/write
Base + \$0002	IIC Control Register 1 (IICC1)	read/write
Base + \$0003	IIC Status Register (IICS)	read
Base + \$0004	IIC Data IO Register (IICD)	read/write
Base + \$0005	IIC Control Register 2 (IICC2)	read/write
Base + \$0006	SMBUS IIC Control and Status Register (IICSMB)	read/write
Base + \$0007	IIC Address Register 2 (IICA2)	read/write
Base + \$0008	IIC SCL Low Time Out Register High (IICSLTH)	read/write
Base + \$0009	IIC SCL Low Time Out Register Low (IICSLTL)	read/write
Base + \$000A	IIC input programmable filter (IICFLT)	read/write

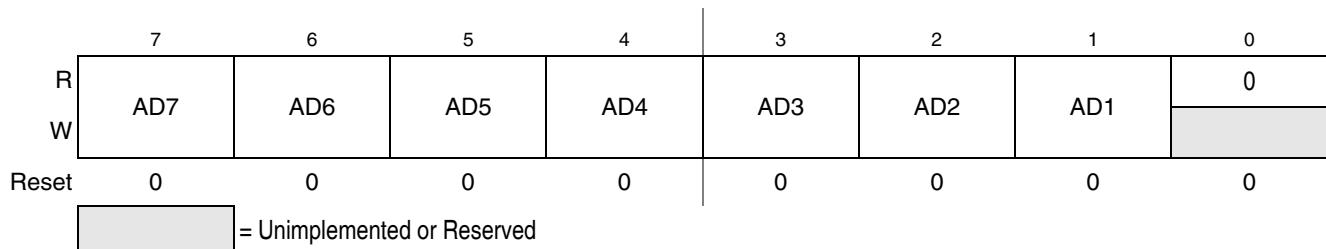
This section consists of the IIC register descriptions in address order.

Refer to the direct-page register summary in the [Chapter 4, “Memory”](#) for the absolute address assignments for all IIC registers. This section refers to registers and control bits only by their names.

#### NOTE

A Freescale-provided equate or header file is used to translate these names into the appropriate absolute addresses.

### 17.3.2 IIC Address Register 1 (IICA1)

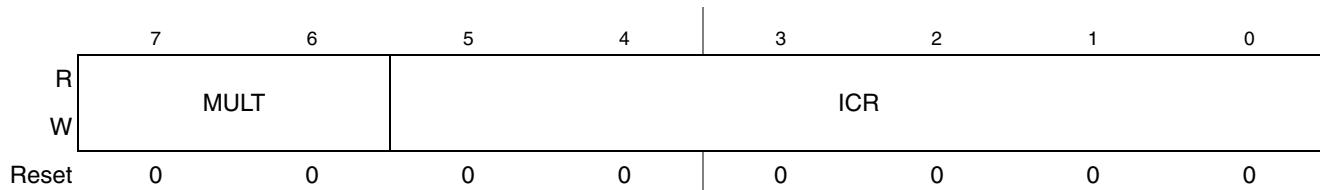


**Figure 17-2. IIC Address Register 1 (IICA1)**

**Table 17-2. IICA1 Field Descriptions**

Field	Description
7:1 AD[7:1]	<b>Slave Address 1</b> — The AD[7:1] field contains the slave address for the IIC module. This field is used on the 7-bit address scheme and the lower seven bits of the 10-bit address scheme.

### 17.3.3 IIC Frequency Divider Register (IICF)

**Figure 17-3. IIC Frequency Divider Register (IICF)****Table 17-3. IICF Field Descriptions**

Field	Description
7:6 MULT	<b>IIC Multiplier Factor</b> — The MULT bits define the multiplier factor mul. This factor is used along with the SCL divider to generate the IIC baud rate. The multiplier factor mul as defined by the MULT bits is provided below. 00 mul = 01 01 mul = 02 10 mul = 04 11 Reserved
5:0 ICR	<b>IIC Clock Rate</b> — The ICR bits are used to prescale the bus clock for bit rate selection. These bits and the MULT bits are used to determine the IIC baud rate, the SDA hold time, the SCL Start hold time and the SCL Stop hold time. <a href="#">Table 17-4</a> provides the SCL divider and hold values for corresponding values of the ICR.  The SCL divider multiplied by multiplier factor mul is used to generate IIC baud rate.  $\text{IIC baud rate} = \text{bus speed (Hz)} / (\text{mul} * \text{SCL divider}) \quad \text{Eqn. 17-1}$ SDA hold time is the delay from the falling edge of SCL (IIC clock) to the changing of SDA (IIC data).  $\text{SDA hold time} = \text{bus period (s)} * \text{mul} * \text{SDA hold value} \quad \text{Eqn. 17-2}$ SCL Start hold time is the delay from the falling edge of SDA (IIC data) while SCL is high (Start condition) to the falling edge of SCL (IIC clock).  $\text{SCL Start hold time} = \text{bus period (s)} * \text{mul} * \text{SCL Start hold value} \quad \text{Eqn. 17-3}$ SCL Stop hold time is the delay from the rising edge of SCL (IIC clock) to the rising edge of SDA (IIC data) while SCL is high (Stop condition).  $\text{SCL Stop hold time} = \text{bus period (s)} * \text{mul} * \text{SCL Stop hold value} \quad \text{Eqn. 17-4}$

For example, if the bus speed is 8 MHz, the table below shows the possible hold time values with different ICR and MULT selections to achieve an IIC baud rate of 100 kbps

MULT	ICR	Hold times ( $\mu$ s)		
		SDA	SCL Start	SCL Stop
0x2	0x00	3.500	3.000	5.500
0x1	0x07	2.500	4.000	5.250
0x1	0x0B	2.250	4.000	5.250
0x0	0x14	2.125	4.250	5.125
0x0	0x18	1.125	4.750	5.125

**Table 17-4. IIC Divider and Hold Values**

<b>ICR (hex)</b>	<b>SCL Divider</b>	<b>SDA Hold Value</b>	<b>SCL Hold (Start) Value</b>	<b>SDA Hold (Stop) Value</b>
00	20	7	6	11
01	22	7	7	12
02	24	8	8	13
03	26	8	9	14
04	28	9	10	15
05	30	9	11	16
06	34	10	13	18
07	40	10	16	21
08	28	7	10	15
09	32	7	12	17
0A	36	9	14	19
0B	40	9	16	21
0C	44	11	18	23
0D	48	11	20	25
0E	56	13	24	29
0F	68	13	30	35
10	48	9	18	25
11	56	9	22	29
12	64	13	26	33
13	72	13	30	37
14	80	17	34	41
15	88	17	38	45
16	104	21	46	53
17	128	21	58	65
18	80	9	38	41
19	96	9	46	49
1A	112	17	54	57
1B	128	17	62	65
1C	144	25	70	73
1D	160	25	78	81
1E	192	33	94	97
1F	240	33	118	121

<b>ICR (hex)</b>	<b>SCL Divider</b>	<b>SDA Hold Value</b>	<b>SCL Hold (Start) Value</b>	<b>SDA Hold (Stop) Value</b>
20	160	17	78	81
21	192	17	94	97
22	224	33	110	113
23	256	33	126	129
24	288	49	142	145
25	320	49	158	161
26	384	65	190	193
27	480	65	238	241
28	320	33	158	161
29	384	33	190	193
2A	448	65	222	225
2B	512	65	254	257
2C	576	97	286	289
2D	640	97	318	321
2E	768	129	382	385
2F	960	129	478	481
30	640	65	318	321
31	768	65	382	385
32	896	129	446	449
33	1024	129	510	513
34	1152	193	574	577
35	1280	193	638	641
36	1536	257	766	769
37	1920	257	958	961
38	1280	129	638	641
39	1536	129	766	769
3A	1792	257	894	897
3B	2048	257	1022	1025
3C	2304	385	1150	1153
3D	2560	385	1278	1281
3E	3072	513	1534	1537
3F	3840	513	1918	1921

### 17.3.4 IIC Control Register (IICC1)

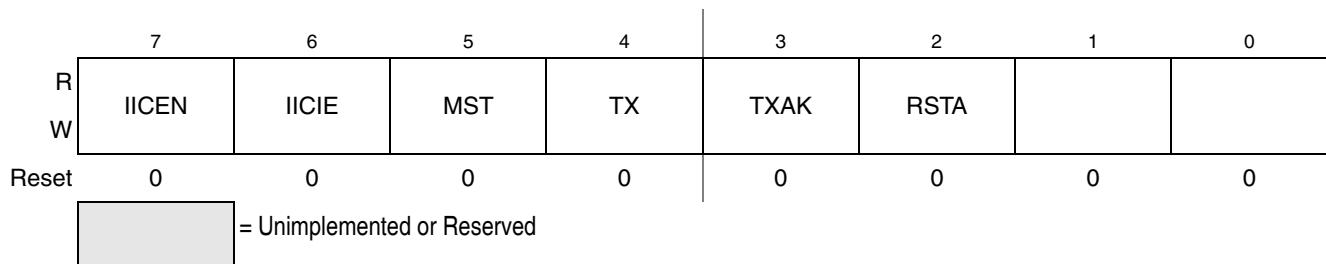


Figure 17-4. IIC Control Register (IICC1)

Table 17-5. IICC1 Field Descriptions

Field	Description
7 IICEN	<b>IIC Enable</b> — The IICEN bit determines whether the IIC module is enabled. 0 IIC is not enabled. 1 IIC is enabled.
6 IICIE	<b>IIC Interrupt Enable</b> — The IICIE bit determines whether an IIC interrupt is requested. 0 IIC interrupt request not enabled. 1 IIC interrupt request enabled.
5 MST	<b>Master Mode Select</b> — When the MST bit is changed from a 0 to a 1, a START signal is generated on the bus and master mode is selected. When this bit changes from a 1 to a 0 a STOP signal is generated and the mode of operation changes from master to slave. 0 Slave mode. 1 Master mode.
4 TX	<b>Transmit Mode Select</b> — The TX bit selects the direction of master and slave transfers. In master mode this bit should be set according to the type of transfer required. Therefore, for address cycles, this bit is always high. When addressed as a slave this bit should be set by software according to the SRW bit in the status register. 0 Receive. 1 Transmit.
3 TXAK	<b>Transmit Acknowledge Enable</b> — This bit specifies the value driven onto the SDA during data acknowledge cycles for both master and slave receivers. The following two conditions effect NAK/ACK generation. If FACK (fast NACK/ACK) is cleared, 0 An acknowledge signal is sent out to the bus on the following receiving data byte. 1 No acknowledge signal response is sent to the bus on the following receiving data byte. If FASK bit is set. no ACK or NACK is sent out after receiving one data byte until this TXAK bit is written 0 An acknowledge signal is sent out to the bus on the current receiving data byte 1 No acknowledge signal response is sent to the bus on the current receiving data byte Note: SCL is held to low until TXAK is written.
2 RSTA (Write Only read always 0)	<b>Repeat START</b> — Writing a 1 to this bit generates a repeated START condition provided it is the current master. Attempting a repeat at the wrong time results in loss of arbitration. 0 No repeat start detected in bus operation. 1 Repeat start generated.

### 17.3.5 IIC Status Register (IICS)

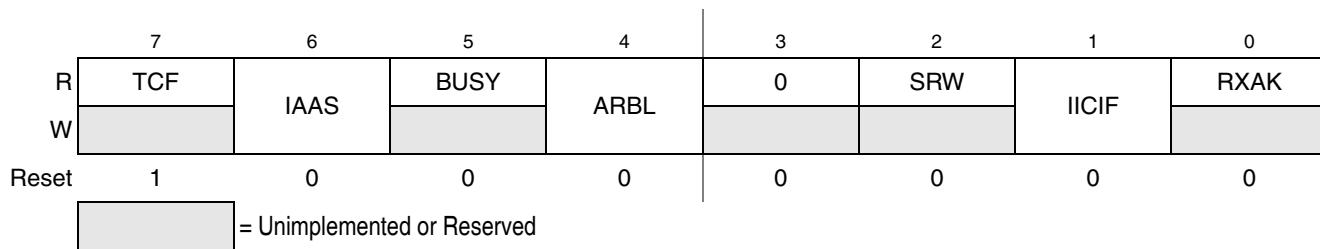


Figure 17-5. IIC Status Register (IICS)

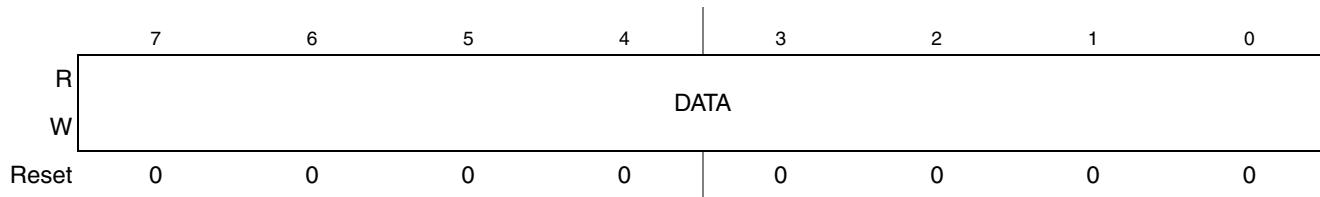
Table 17-6. IICS Field Descriptions

Field	Description
7 TCF	<b>Transfer Complete Flag</b> — This bit is set on the completion of a byte and acknowledge bit transfer. Note that this bit is only valid during or immediately following a transfer to the IIC module or from the IIC module. The TCF bit is cleared by reading the IICD register in receive mode or writing to the IICD in transmit mode. 0 Transfer in progress. 1 Transfer complete.
6 IAAS	<b>Addressed as a Slave</b> — The IAAS bit is set when one of the following conditions is met 1) When the calling address matches the programmed slave address, 2) If the GCAEN bit is set and a general call is received. 3) If SIICAEN bit is set, when the calling address matches the second programmed slave address 4) If ALERTEN bit is set and SMBus alert response address is received  This bit is set before ACK bit. The CPU needs to check the SRW bit and set TX/RX bit accordingly. Writing the IICC1 register with any value clears this bit. 0 Not addressed. 1 Addressed as a slave.
5 BUSY	<b>Bus Busy</b> — The BUSY bit indicates the status of the bus regardless of slave or master mode. The BUSY bit is set when a START signal is detected and cleared when a STOP signal is detected. 0 Bus is idle. 1 Bus is busy.
4 ARBL	<b>Arbitration Lost</b> — This bit is set by hardware when the arbitration procedure is lost. The ARBL bit must be cleared by software, by writing a 1 to it. 0 Standard bus operation. 1 Loss of arbitration.
2 SRW	<b>Slave Read/Write</b> — When addressed as a slave the SRW bit indicates the value of the R/W command bit of the calling address sent to the master. 0 Slave receive, master writing to slave. 1 Slave transmit, master reading from slave.

**Table 17-6. IICS Field Descriptions (continued)**

Field	Description
1 IICIF	<b>IIC Interrupt Flag</b> — The IICIF bit is set when an interrupt is pending. This bit must be cleared by software, by writing a 1 to it in the interrupt routine. One of the following events can set the IICIF bit: <ul style="list-style-type: none"> <li>• One byte transfer including ACK/NACK bit completes if FACK = 0</li> <li>• One byte transfer including ACK/NACK bit completes if FACK = 1 and this byte is an address byte</li> <li>• One byte transfer excluding ACK/NACK bit completes if FACK = 1 and this byte is a data byte. an ACK or NACK is sent out on the bus by writing 0 or 1 to TXAK after this bit is set.</li> <li>• Match of slave addresses to calling address (Primary Slave address, General Call address, Alert Response address, and Second Slave address) (Received address is stored in data register)</li> <li>• Arbitration lost</li> <li>• Timeouts in SMBus mode except high timeout</li> </ul> 0 No interrupt pending. 1 Interrupt pending.
0 RXAK	<b>Receive Acknowledge</b> — When the RXAK bit is low, it indicates an acknowledge signal is received after the completion of one byte of data transmission on the bus. If the RXAK bit is high it means that no acknowledge signal is detected. 0 Acknowledge received. 1 No acknowledge received.

### 17.3.6 IIC Data I/O Register (IICD)

**Figure 17-6. IIC Data I/O Register (IICD)****Table 17-7. IICD Field Descriptions**

Field	Description
7:0 DATA	<b>Data</b> — In master transmit mode, when data is written to the IICD, a data transfer is initiated. The most significant bit is sent first. In master receive mode, reading this register initiates receiving of the next byte of data.

#### NOTE

When transitioning out of master receive mode, the IIC mode should be switched before reading the IICD register to prevent an inadvertent initiation of a master receive data transfer.

In slave mode, the same functions are available after an address match has occurred.

The TX bit in IICC must correctly reflect the desired direction of transfer in master and slave modes for the transmission to begin. For example, if the IIC is configured for master transmit but a master receive is desired, then reading the IICD does not initiate the receive.

Reading the IICD returns the last byte received while the IIC is configured in either master receive or slave receive modes. The IICD does not reflect every byte that is transmitted on the IIC bus, nor can software verify that a byte has been written to the IICD correctly by reading it back.

In master transmit mode, the first byte of data written to IICD following assertion of MST (Start bit) or assertion of RSTA bit (repeated Start ) is used for the address transfer and should comprise of the calling address (in bit 7 to bit 1) concatenated with the required R/W bit (in position bit 0).

### 17.3.7 IIC Control Register 2 (IICC2)

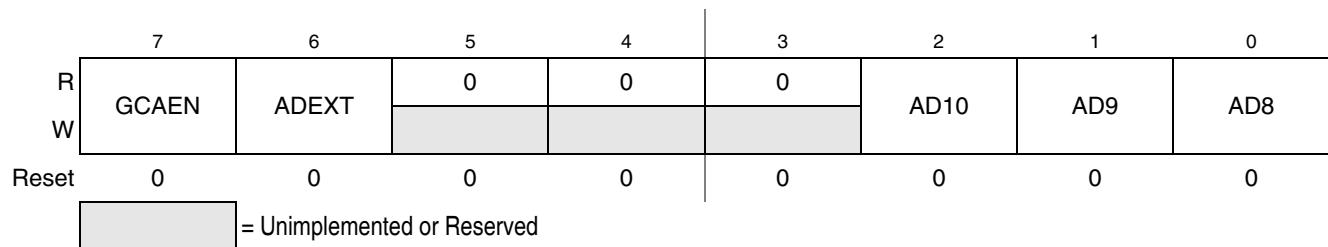


Figure 17-7. IIC Control Register (IICC2)

Table 17-8. IICC2 Field Descriptions

Field	Description
7 GCAEN	<b>General Call Address Enable</b> — The GCAEN bit enables or disables general call address. 0 General call address is disabled 1 General call address is enabled.
6 ADEXT	<b>Address Extension</b> — The ADEXT bit controls the number of bits used for the slave address. 0 7-bit address scheme 1 10-bit address scheme
2:0 AD[10:8]	<b>Slave Address</b> — The AD[10:8] field contains the upper three bits of the slave address in the 10-bit address scheme. This field is only valid when the ADEXT bit is set.

### 17.3.8 IIC SMBus Control and Status Register (IICSMB)

	7	6	5	4	3	2	1	0
R W	FACK	ALERTEN	SIICAEN	TCKSEL	SLTF	SHTF		
Reset	0	0	0	0	0	0	0	0
	= Unimplemented or Reserved							

Table 17-9. IICSMB Field Descriptions

Field	Description
7 FACK	<b>Fast NACK/ACK enable</b> — For SMBus Packet Error Checking, CPU should be able to issue an ACK or NACK according to the result of receiving data byte. 0 ACK or NACK is sent out on the following receiving data byte. 1 Writing an 0 to TXAK after receiving data byte generates an ACK; Writing an 1 to TXAK after receiving data byte generates a NACK
6 ALERTEN reserved	<b>SMBus Alert Response Address Enable</b> — The ALERTEN bit enables or disable SMBus alert response address. 0 SMBus alert response address matching is disabled 1 SMBus alert response address matching is enabled.
5 SIICAEN	<b>Second IIC Address Enable</b> — The SIICAEN bit enables or disable SMBus device default address. 0 IIC Address Register 2 matching is disabled. 1 IIC Address Register 2 matching is enabled.
4 TCKSEL	<b>Time Out Counter Clock Select</b> — This bit selects the clock sources of Time Out Counter 0 Time Out Counter counts at bus/64 frequency 1 Time Out Counter counts at the bus frequency
3 SLTF	<b>SCL Low Timeout Flag</b> — This read-only bit is set to logic 1 when IICSLT loaded non zero value (LoValue) and a SCL Low Time Out occurs. This bit is cleared by software, by writing a logic 1 to it 0 No LOW TIME OUT occurs. 1 A LOW TIME OUT occurs. Note: LOW TIME OUT function is disabled when IIC SCL LOW TIMER OUT register is set to zero.
2 SHTF	<b>SCL High Timeout Flag</b> — This read-only bit is set to logic 1 when SCL and SDA are held high more than clock * LoValue/512, which indicates the Bus Free. This bit is cleared automatically. 0 No HIGH TIMEOUT occurs. 1 An HIGH TIMEOUT occurs.

#### NOTE

1. A master can assume that the bus is free if it detects that the clock and data signals are high for greater than THIGH,MAX, however, the SHTF rises in bus transmission process but bus idle state.
2. When TCKSEL=1 there is no meaning to monitor SHTF since the bus speed is too high to match the protocol of SMBus.

### 17.3.9 IIC Address Register 2 (IICA2)

	7	6	5	4		3	2	1	0
R	SAD7	SAD6	SAD5	SAD4	SAD3	SAD2	SAD1	0	
W									
Reset	1	1	0	0	0	0	1	0	

= Unimplemented or Reserved

Field	Description
7:1 SAD[7:1]	<b>SMBus Address</b> — The AD[7:1] field contains the slave address to be used by the SMBus. This field is used on the device default address or other related address.

### 17.3.10 IIC SCL Low Time Out Register High (IICSLTH)

	7	6	5	4		3	2	1	0
R	SSLT15	SSLT14	SSLT13	SSLT12	SSLT11	SSLT10	SSLT9	SSLT8	
W									
Reset	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Field	Description
7:0 SSLT[15:8]	The value in this register is the most significant byte of SCL low time out value that determines the time-out period of SCL low.

### 17.3.11 IIC SCL LowTime Out register Low (IICSLTL)

	7	6	5	4		3	2	1	0
R	SSLT7	SSLT6	SSLT5	SSLT4	SSLT3	SSLT2	SSLT1	SSLT0	
W									
Reset	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Field	Description
7:0 SSLT[7:0]	The value in this register is the least significant byte of SCL low time out value that determines the time-out period of SCL low..

### 17.3.12 IIC Programmable Input Glitch Filter (IICFLT)

	7	6	5	4		3	2	1	0
R	0	0	0	0		FLT3	FLT2	FLT1	FLT0
W						0	0	0	0
Reset	0	0	0	0		0	0	0	0
	= Unimplemented or Reserved								

**Table 17-10. IICFLT Field Descriptions**

Field	Description
3:0 FLT	<p>IIC Programmable Filter Factor contains the programming controls for the width of glitch (in terms of bus clock cycles) the filter should absorb; in other words, the filter does not let glitches less than or equal to this width setting pass. FLT[3:0]</p> <p>0000 No Filter / Bypass      0001 Filter glitches up to width of 1 (half) IPBUS clock cycle      0010 Filter glitches up to width of 2 (half) IPBUS clock cycles      0011 Filter glitches up to width of 3 (half) IPBUS clock cycles      0100 Filter glitches up to width of 4 (half) IPBUS clock cycles      0101 Filter glitches up to width of 5 (half) IPBUS clock cycles      0110 Filter glitches up to width of 6 (half) IPBUS clock cycles      0111 Filter glitches up to width of 7 (half) IPBUS clock cycles      1000 Filter glitches up to width of 8 (half) IPBUS clock cycle      1001 Filter glitches up to width of 9 (half) IPBUS clock cycles      1010 Filter glitches up to width of 10 (half) IPBUS clock cycles      1011 Filter glitches up to width of 11 (half) IPBUS clock cycles      1100 Filter glitches up to width of 12 (half) IPBUS clock cycles      1101 Filter glitches up to width of 13 (half) IPBUS clock cycles      1110 Filter glitches up to width of 14 (half) IPBUS clock cycles      1111 Filter glitches up to width of 15 (half) IPBUS clock cycles</p> <p>NOTE: If the filter input clock is connected to 2X IPBUS clock, then the (half) affects the width of glitches</p>

## 17.4 Functional Description

This section provides a complete functional description of the IIC module.

### 17.4.1 IIC Protocol

The IIC bus system uses a serial data line (SDA) and a serial clock line (SCL) for data transfer. All devices connected to it must have open drain or open collector outputs. A logic AND function is exercised on both lines with external pull-up resistors. The value of these resistors is system dependent.

Normally, a standard communication is composed of four parts:

- START signal
- Slave address transmission
- Data transfer
- STOP signal

The STOP signal should not be confused with the CPU STOP instruction. The IIC bus system communication is described briefly in the following sections and illustrated in [Figure 17-8](#).

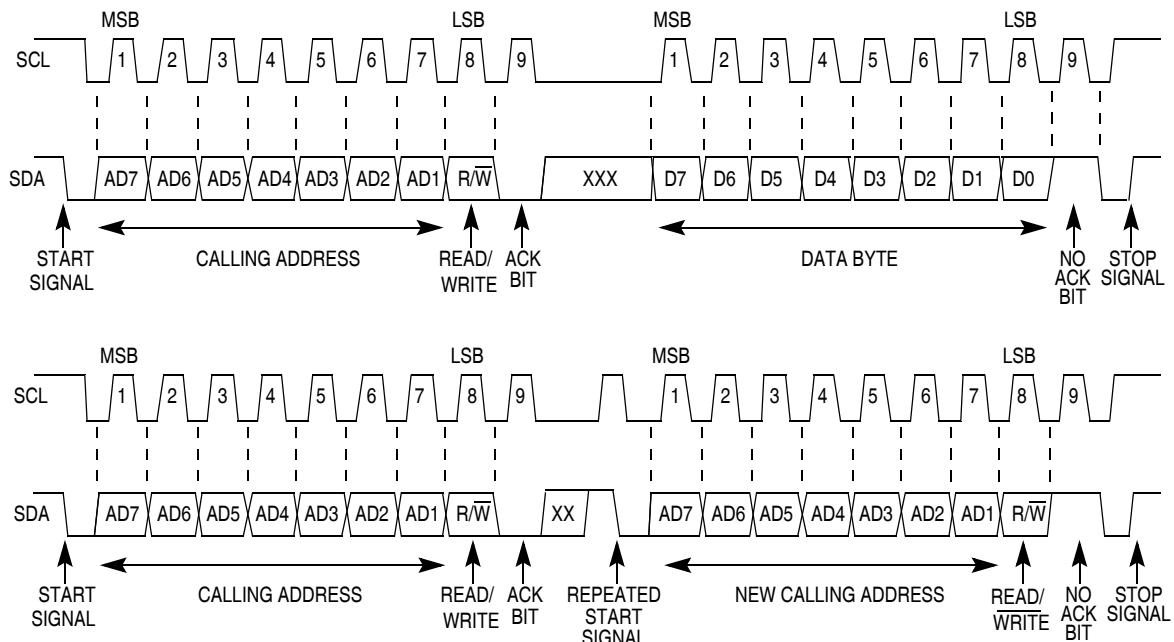


Figure 17-8. IIC Bus Transmission Signals

### 17.4.1.1 START Signal

When the bus is free; i.e., no master device is engaging the bus (both SCL and SDA lines are at logical high), a master may initiate communication by sending a START signal. As shown in [Figure 17-8](#), a START signal is defined as a high-to-low transition of SDA while SCL is high. This signal denotes the beginning of a new data transfer (each data transfer may contain several bytes of data) and brings all slaves out of their idle states.

### 17.4.1.2 Slave Address Transmission

The first byte of data transferred immediately after the START signal is the slave address transmitted by the master. This is a seven-bit calling address followed by a R/W bit. The R/W bit tells the slave the desired direction of data transfer.

- 1 = Read transfer, the slave transmits data to the master.
- 0 = Write transfer, the master transmits data to the slave.

Only the slave with a calling address that matches the one transmitted by the master responds by sending back an acknowledge bit. This is done by pulling the SDA low at the ninth clock (see [Figure 17-8](#)).

No two slaves in the system may have the same address. If the IIC module is the master, it must not transmit an address that is equal to its own slave address. The IIC cannot be master and slave at the same time. However, if arbitration is lost during an address cycle, the IIC reverts to slave mode and operate correctly even if it is being addressed by another master.

### 17.4.1.3 Data Transfer

Before successful slave addressing is achieved, the data transfer can proceed byte-by-byte in a direction specified by the R/W bit sent by the calling master.

All transfers that come after an address cycle are referred to as data transfers, even if they carry sub-address information for the slave device

Each data byte is 8 bits long. Data may be changed only while SCL is low and must be held stable while SCL is high as shown in [Figure 17-8](#). There is one clock pulse on SCL for each data bit, the MSB being transferred first. Each data byte is followed by a ninth (acknowledge) bit, which is signalled from the receiving device. An acknowledge is signalled by pulling the SDA low at the ninth clock. In summary, one complete data transfer needs nine clock pulses.

If the slave receiver does not acknowledge the master in the ninth bit time, the SDA line must be left high by the slave. The master interprets the failed acknowledge as an unsuccessful data transfer.

If the master receiver does not acknowledge the slave transmitter after a data byte transmission, the slave interprets this as an end of data transfer and releases the SDA line.

In either case, the data transfer is aborted and the master does one of two things:

- Relinquishes the bus by generating a STOP signal.
- Commences a new calling by generating a repeated START signal.

#### 17.4.1.4 STOP Signal

The master can terminate the communication by generating a STOP signal to free the bus. However, the master may generate a START signal followed by a calling command without generating a STOP signal first. This is called repeated START. A STOP signal is defined as a low-to-high transition of SDA while SCL at logical 1 (see [Figure 17-8](#)).

The master can generate a STOP even if the slave has generated an acknowledge at which point the slave must release the bus.

#### 17.4.1.5 Repeated START Signal

As shown in [Figure 17-8](#), a repeated START signal is a START signal generated without first generating a STOP signal to terminate the communication. This is used by the master to communicate with another slave or with the same slave in different mode (transmit/receive mode) without releasing the bus.

#### 17.4.1.6 Arbitration Procedure

The IIC bus is a true multi-master bus that allows more than one master to be connected on it. If two or more masters try to control the bus at the same time, a clock synchronization procedure determines the bus clock, for which the low period is equal to the longest clock low period and the high is equal to the shortest one among the masters. The relative priority of the contending masters is determined by a data arbitration procedure, a bus master loses arbitration if it transmits logic 1 while another master transmits logic 0. The losing masters immediately switch over to slave receive mode and stop driving SDA output. In this case, the transition from master to slave mode does not generate a STOP condition. Meanwhile, a status bit is set by hardware to indicate loss of arbitration.

#### 17.4.1.7 Clock Synchronization

Because wire-AND logic is performed on the SCL line, a high-to-low transition on the SCL line affects all the devices connected on the bus. The devices start counting their low period and after a device's clock has gone low, it holds the SCL line low until the clock high state is reached. However, the change of low to high in this device clock may not change the state of the SCL line if another device clock is still within its low period. Therefore, synchronized clock SCL is held low by the device with the longest low period. Devices with shorter low periods enter a high wait state during this time (see [Figure 17-9](#)). When all devices concerned have counted off their low period, the synchronized clock SCL line is released and pulled high. There is then no difference between the device clocks and the state of the SCL line and all the devices start counting their high periods. The first device to complete its high period pulls the SCL line low again.

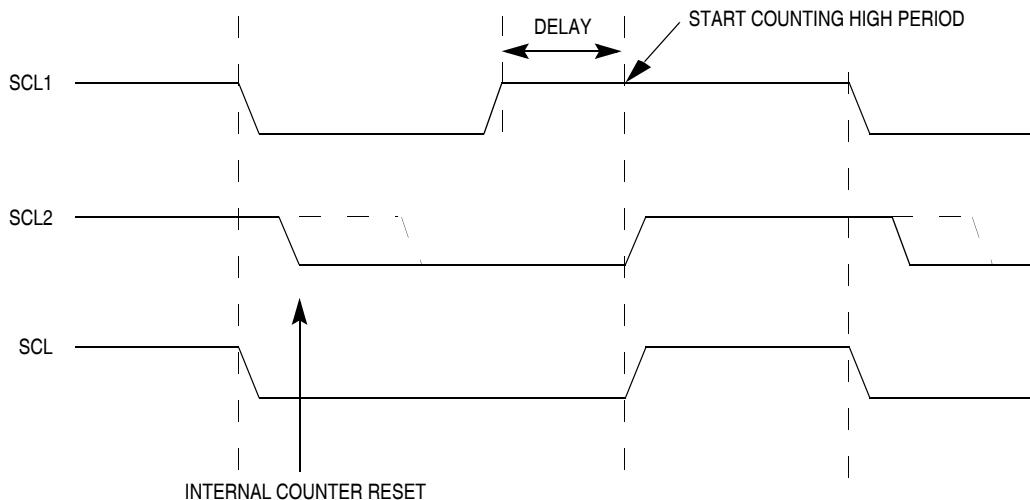


Figure 17-9. IIC Clock Synchronization

#### 17.4.1.8 Handshaking

The clock synchronization mechanism can be used as a handshake in data transfer. Slave devices may hold the SCL low after completion of one byte transfer (9 bits). In such case, it halts the bus clock and forces the master clock into wait states until the slave releases the SCL line.

#### 17.4.1.9 Clock Stretching

The clock synchronization mechanism can be used by slaves to slow down the bit rate of a transfer. After the master has driven SCL low the slave can drive SCL low for the required period and then release it. If the slave SCL low period is greater than the master SCL low period then the resulting SCL bus signal low period is stretched.

## 17.4.2 10-bit Address

For 10-bit addressing, 11110b is used for the first 5 bits of the first address byte. Various combinations of read/write formats are possible within a transfer that includes 10-bit addressing.

### 17.4.2.1 Master-Transmitter Addresses a Slave-Receiver

The transfer direction is not changed (see [Table 17-11](#)). When a 10-bit address follows a START condition, each slave compares the first seven bits of the first byte of the slave address (11110XX) with its own address and tests whether the eighth bit (R/W direction bit) is 0. It is possible that more than one device finds a match and generates an acknowledge (A1). Each slave that finds a match compares the eight bits of the second byte of the slave address with its own address, but only one slave finds a match and generate an acknowledge (A2). The matching slave remains addressed by the master until it receives a STOP condition (P) or a repeated START condition (Sr) followed by a different slave address.

S	Slave Address 1st 7 bits 11110 + AD10 + AD9	R/W 0	A1	Slave Address 2nd byte AD[8:1]	A2	Data	A	...	Data	A/A	P
---	--	----------	----	-----------------------------------	----	------	---	-----	------	-----	---

**Table 17-11. Master-Transmitter Addresses Slave-Receiver with a 10-bit Address**

After the master-transmitter has sent the first byte of the 10-bit address, the slave-receiver see an IIC interrupt. User software must ensure that for this interrupt, the contents of IICD are ignored and not treated as valid data.

### 17.4.2.2 Master-Receiver Addresses a Slave-Transmitter

The transfer direction is changed after the second R/W bit (see [Table 17-12](#)). Up to and including acknowledge bit A2, the procedure is the same as that described for a master-transmitter addressing a slave-receiver. After the repeated START condition (Sr), a matching slave remembers that it was addressed before. This slave then checks whether the first seven bits of the first byte of the slave address following Sr are the same as they were after the START condition (S), and tests whether the eighth (R/W) bit is 1. If there is a match, the slave considers that it is addressed as a transmitter and generates acknowledge A3. The slave-transmitter remains addressed until it receives a STOP condition (P) or a repeated START condition (Sr) followed by a different slave address.

After a repeated START condition (Sr), all other slave devices also compare the first seven bits of the first byte of the slave address with their own addresses and test the eighth (R/W) bit. However, none of them are addressed because R/W = 1 (for 10-bit devices), or the 11110XX slave address (for 7-bit devices) does not match.

S	Slave Address 1st 7 bits 11110 + AD10 + AD9	R/W 0	A1	Slave Address 2nd byte AD[8:1]	A2	Sr	Slave Address 1st 7 bits 11110 + AD10 + AD9	R/W 1	A3	Data	A	...	Data	A	P
---	--	----------	----	-----------------------------------	----	----	--	----------	----	------	---	-----	------	---	---

**Table 17-12. Master-Receiver Addresses a Slave-Transmitter with a 10-bit Address**

After the master-receiver has sent the first byte of the 10-bit address, the slave-transmitter see an IIC interrupt. User software must ensure that for this interrupt, the contents of IICD are ignored and not treated as valid data.

### 17.4.3 Address Matching

All received Addresses can be requested in 7-bit or 10-bit address. IIC Address Register 1, which contains IIC primary slave address, always participates the address matching process. If the GCAEN bit is set, general call participates the address matching process. If the ALERTEN bit is set, alert response participates the address matching process. If SIICAEN bit is set, the IIC Address Register 2 participates the address matching process.

When the IIC responds to one of above mentioned address, it acts as a slave-receiver and the IAAS bit is set after the address cycle. Software need to read the IICD register after the first byte transfer to determine which the address is matched.

### 17.4.4 System Management Bus Specification

SMBus provides a control bus for system and power management related tasks. A system may use SMBus to pass messages to and from devices instead of tripping individual control lines. Removing the individual control lines reduces pin count. Accepting messages ensures future expandability. With System Management Bus, a device can provide manufacturer information, tell the system what its model/part number is, save its state for a suspend event, report different types of errors, accept control parameters, and return its status.

#### 17.4.4.1 Timeouts

The TTIMEOUT,MIN parameter allows a master or slave to conclude that a defective device is holding the clock low indefinitely or a master is intentionally trying to drive devices off the bus. It is highly recommended that a slave device release the bus (stop driving the bus and let SCL and SDA float high) when it detects any single clock held low longer than TTIMEOUT,MIN. Devices that have detected this condition should reset their communication and be able to receive a new START condition in no later than TTIMEOUT,MAX.

SMBus defines a clock low time-out, TTIMEOUT of 35 ms and specifies TLOW: SEXT as the cumulative clock low extend time for a slave device and specifies TLOW: MEXT as the cumulative clock low extend time for a master device.

##### 17.4.4.1.1 SCL Low Timeout

If the SCL line is held low by a slave device on the bus, no further communication is possible. Furthermore, the master cannot force the SCL line high to correct the error condition. To solve this problem, the SMBus protocol specifies that devices participating in a transfer must detect any clock cycle held low longer than a “timeout” value condition. Devices that have detected the timeout condition must reset the communication. When active master, if the IIC detects that SMBCLK low has exceeded the value of TTIMEOUT,MIN it must generate a stop condition within or after the current data byte in the transfer process. When slave, upon detection of the TTIMEOUT,MIN condition, the IIC shall reset its communication and be able to receive a new START condition.

#### 17.4.4.1.2 SCL High (SMBus Free) Timeout

The IIC shall assume that the bus is idle, when it has determined that the SMBCLK and SMBDAT signals are high for at least THIGH:MAX. HIGH timeout can occur in two ways:

1. HIGH timeout detected after a STOP condition appears on the bus.
2. HIGH timeout detected after a START condition, but before a STOP condition appears on the bus.

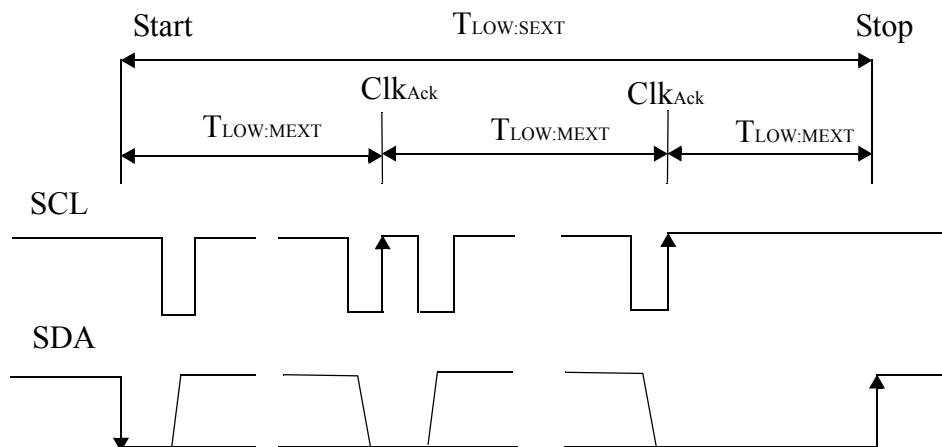
Any master detecting either scenario can assume the bus is free then SHTF rises. HIGH timeout occurred in scenario 2 if it ever detects that both conditions are true: BUSY bit is high and SHTF is high.

#### 17.4.4.1.3 CSMBCLK TIMEOUT MEXT

[Figure 17-10](#) illustrates the definition of the timeout intervals, TLOW:SEXT and TLOW:MEXT. When master mode, the I2C must not cumulatively extend its clock cycles for a period greater than TLOW:MEXT within a byte, where each byte is defined as START-to-ACK, ACK-to-ACK, or ACK-to-STOP. When CSMBCLK TIMEOUT MEXT occurs, SMBus MEXT rises and also triggers the SLTF.

#### 17.4.4.1.4 CSMBCLK TIMEOUT SEXT

A Master is allowed to abort the transaction in progress to any slave that violates the TLOW:SEXT or TTIMEOUT,MIN specifications. This can be accomplished by the Master issuing a STOP condition at the conclusion of the byte transfer in progress. When slave, the I2C must not cumulatively extend its clock cycles for a period greater than TLOW:SEXT during any message from the initial START to the STOP. When CSMBCLK TIMEOUT SEXT occurs, SEXT rises and also triggers SLTF.



**Figure 17-10. Timeout measurement intervals**

Note: CSMBCLK TIMEOUT SEXT and MEXT are optional functions that are implemented in second step.

#### 17.4.4.2 FAST ACK and NACK

To improve reliability and communication robustness, implementation of Packet Error Checking (PEC) by SMBus devices is optional for SMBus devices but required for devices participating in and only during the Address Resolution Protocol (ARP) process. The PEC is a CRC-8 error checking byte, calculated on all the message bytes. The PEC is appended to the message by the device that supplied the last data byte. If the PEC is present but not correct, a NACK is issued by receiver. Otherwise, an ACK is issued. In order to calculate the CRC-8 by software, this module can hold SCL line to low after receiving eighth SCL (bit 8th) if this byte is a data byte. So software can determine whether an ACK or NACK should be sent out to the bus by setting or clearing TXAK bit if FASK (fast ACK/NACK enable bit) is enabled.

SMBus requires devices to acknowledge their own address always, as a mechanism to detect a removable devices presence on the bus (battery, docking station, etc.). Besides to indicate a slave device busy condition, SMBus is using the NACK mechanism also to indicate the reception of an invalid command or data. Since such a condition may occur on the last byte of the transfer, it is required that SMBus devices have the ability to generate the not acknowledge after the transfer of each byte and before the completion of the transaction. This is important because SMBus does not provide any other resend signaling. This difference in the use of the NACK signaling has implications on the specific implementation of the SMBus port, especially in devices that handle critical system data such as the SMBus host, and the SBS components.

#### NOTE

In the last byte of master receive slave transmit mode, the master should send NACK to bus so FACK should be switched off before the last byte transmit.

### 17.5 Resets

The IIC is disabled after reset. The IIC cannot cause an MCU reset.

### 17.6 Interrupts

The IIC generates a single interrupt.

An interrupt from the IIC is generated when any of the events in [Table 17-13](#) occur, provided the IICIE bit is set. The interrupt is driven by bit IICIF (of the IIC status register) and masked with bit IICIE (of the IIC control register). The IICIF bit must be cleared by software by writing a 1 to it in the interrupt routine. Determine the interrupt type by reading the status register. For SMBus timeouts interrupt, the interrupt is driven by SLTF and masked with bit IICIE. The SLTF bit must be cleared by software by writing a 1 to it in the interrupt routine. Determine the interrupt type by reading the status register.

**NOTE**

In Master receive mode the FACK should be set zero before the last byte transfer.

**Table 17-13. Interrupt Summary**

Interrupt Source	Status	Flag	Local Enable
Complete 1-byte transfer	TCF	IICIF	IICIE
Match of received calling address	IAAS	IICIF	IICIE
Arbitration Lost	ARBL	IICIF	IICIE
SMBus Timeout Interrupt Flag	SLTF	IICIF	IICIE

**17.6.1 Byte Transfer Interrupt**

The TCF (transfer complete flag) bit is set at the falling edge of the ninth clock to indicate the completion of byte transfer.

**17.6.2 Address Detect Interrupt**

When the calling address matches the programmed slave address (IIC address register) or when the GCAEN bit is set and a general call is received, the IAAS bit in the status register is set. The CPU is interrupted, provided the IICIE is set. The CPU must check the SRW bit and set its Tx mode accordingly.

**17.6.3 Arbitration Lost Interrupt**

The IIC is a true multi-master bus that allows more than one master to be connected on it. If two or more masters try to control the bus at the same time, the relative priority of the contending masters is determined by a data arbitration procedure. The IIC module asserts this interrupt when it loses the data arbitration process and the ARBL bit in the status register is set.

Arbitration is lost in the following circumstances:

- SDA sampled as a low when the master drives a high during an address or data transmit cycle.
- SDA sampled as a low when the master drives a high during the acknowledge bit of a data receive cycle.
- A START cycle is attempted when the bus is busy.
- A repeated START cycle is requested in slave mode.
- A STOP condition is detected when the master did not request it.

This bit must be cleared by software by writing a 1 to it.

**17.6.4 Timeouts Interrupt in SMBus**

When IICIE is set, the IIC asserts a timeout interrupt output SLTF upon detection of any of the mentioned timeout conditions, with one exception. The HIGH TIMEOUT mechanism shall not be used to influence the timeout interrupt output, because the HIGH TIMEOUT indicates an idle condition on the bus. SLTF rises when it matches the HIGH TIMEOUT and fall automatically to just indicate the bus status.

### 17.6.5 Programmable input glitch filter

An IIC glitch filter are added outside the IIC legacy modules, but within the IIC package. This filter can absorb glitches on the IIC clock and data lines for I2C module. The width of the glitch to absorb can be specified in terms of number of half bus clock cycles. A single glitch filter control register is provided as IICFLT. Effectively, any down-up-down or up-down-up transition on the data line that occurs within the number of clock cycles programmed here is ignored by the IIC. The programmer only needs to specify the size of glitch (in terms of bus clock cycles) for the filter to absorb and not pass.

## 17.7 Initialization/Application Information

### Module Initialization (Slave)

1. Write: IICC2
  - to enable or disable general call
  - to select 10-bit or 7-bit addressing mode
2. Write: IICA1
  - to set the slave address
3. Write: IICC1
  - to enable IIC and interrupts
4. Initialize RAM variables (IICEN = 1 and IICIE = 1) for transmit data
5. Initialize RAM variables used to achieve the routine shown in [Figure 17-11](#)

### Module Initialization (Master)

1. Write: IICF
  - to set the IIC baud rate (example provided in this chapter)
2. Write: IICC1
  - to enable IIC and interrupts
3. Initialize RAM variables (IICEN = 1 and IICIE = 1) for transmit data
4. Initialize RAM variables used to achieve the routine shown in [Figure 17-11](#)
5. Write: IICC1
  - to enable TX
6. Write: IICC1
  - to enable MST (master mode)
7. Write: IICD
  - with the address of the target slave. (The LSB of this byte determines whether the communication is master receive or transmit.)

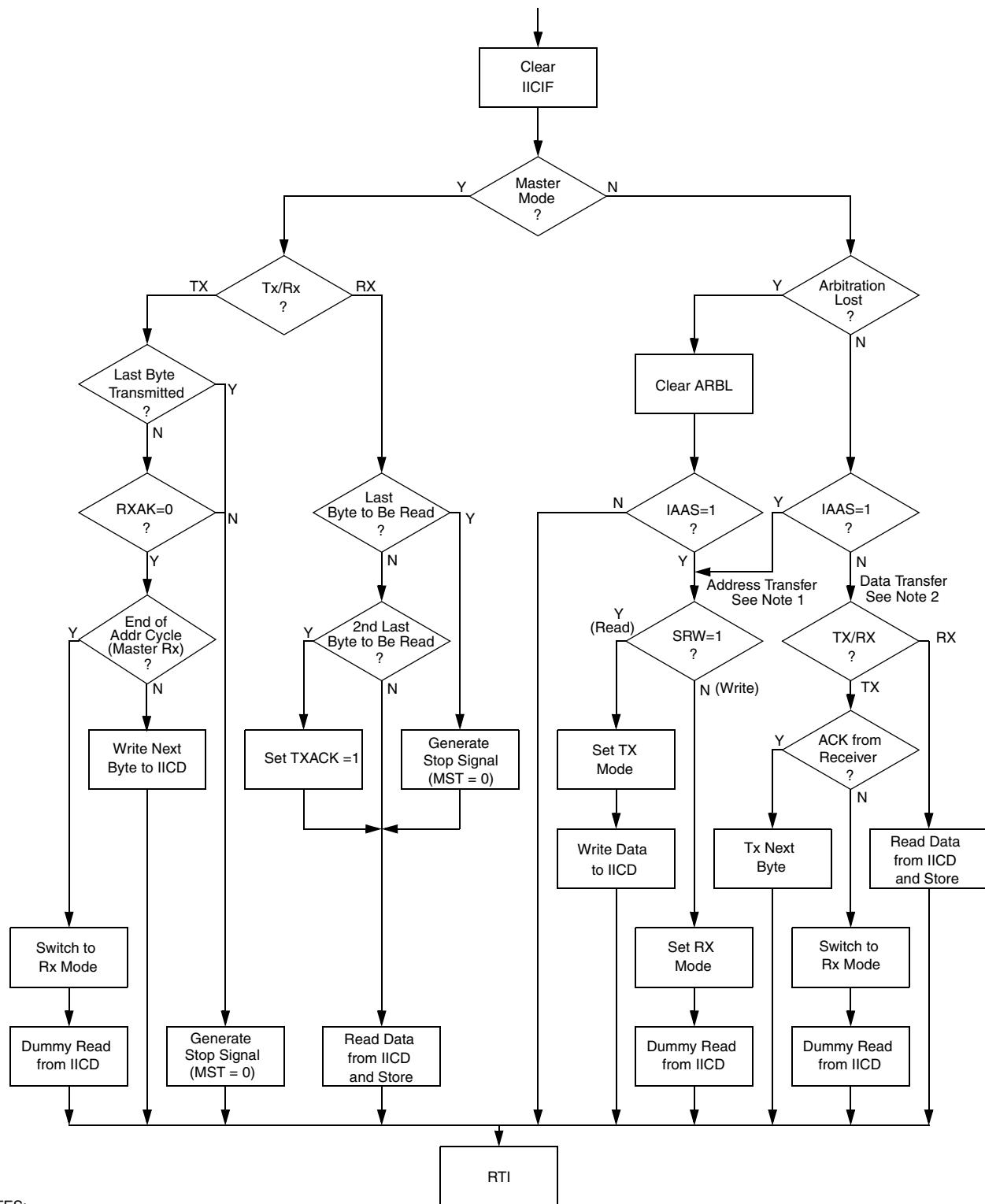
### Module Use

The routine shown in [Figure 17-11](#) can handle both master and slave IIC operations. For slave operation, an incoming IIC message that contains the proper address begins IIC communication. For master operation, communication must be initiated by writing to the IICD register.

### Register Model

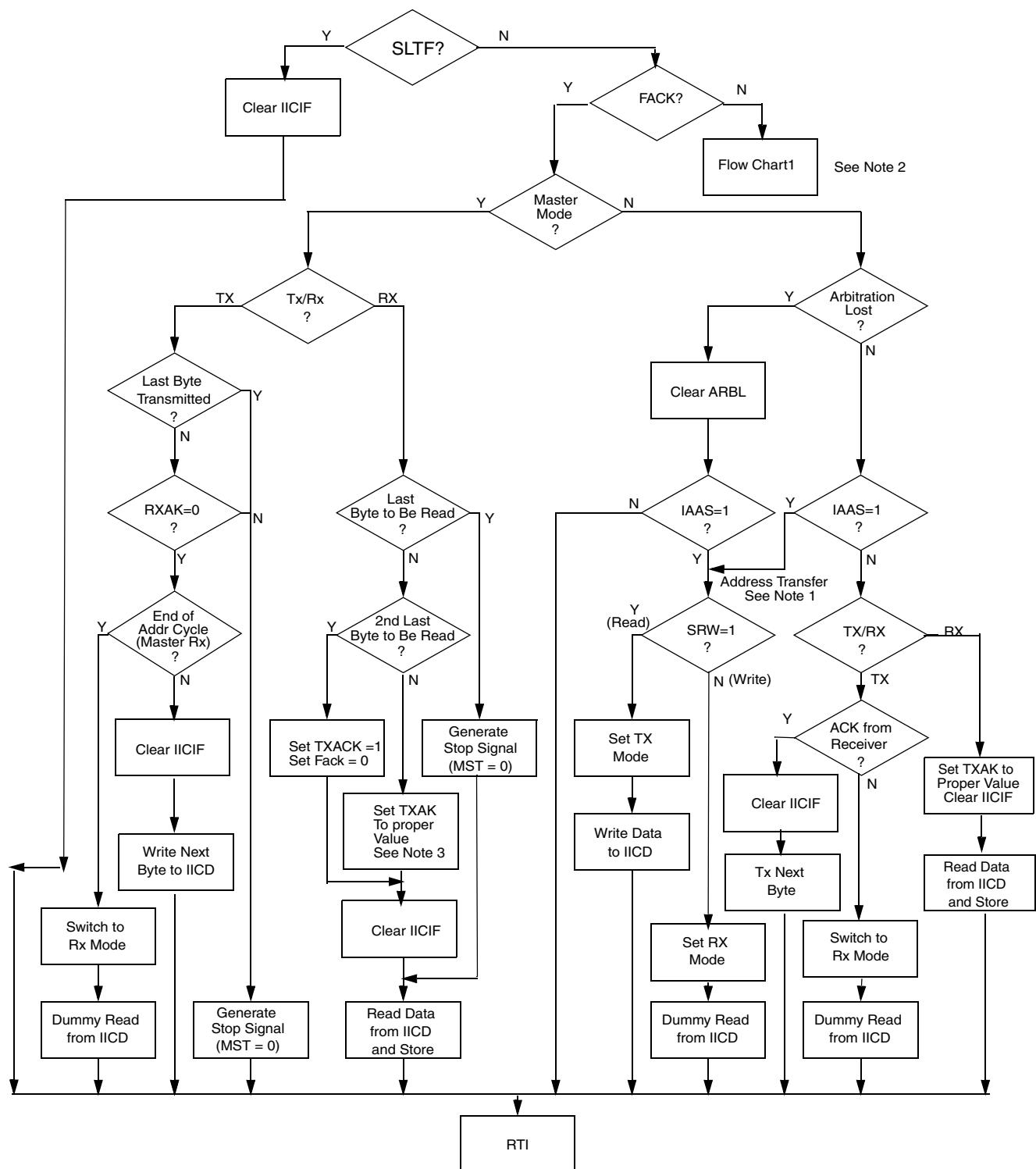
IICA1	AD[7:1] 0						
Address to which the module responds when addressed as a slave (in slave mode)							
IICF	MULT	ICR					
Baud rate = BUSCLK / (2 x MULT x (SCL DIVIDER))							
IICC1	IICEN	IICIE	MST	TX	TXAK	RSTA	0 0
Module configuration							
IICS	TCF	IAAS	BUSY	ARBL	0	SRW	IICIF RXAK
Module status flags							
IICD	DATA						
Data register; Write to transmit IIC data read to read IIC data							
IICC2	GCAEN	ADEXT	0	0	0	AD10	AD9 AD8
Address configuration							
IICSMBS	FACK	ALERTEN	SIICAEN	TCKSEL	SLTF	SLHF	0 0
IIC SMBus Control and Status Register							
IICA2	SAD[7:1] 0						

IICSLTH	SSLT[15:8]							
	IIC SCL Low Time Out Register High							
IICSLTL	SSLT[7:0]							
	IIC SCL Low Time Out Register Low							
IICFLT	0	0	0	0	FLT3	FLT2	FLT1	FLT0
	IIC Programmable Input Glitch Filter							



## NOTES:

- If general call is enabled, a check must be done to determine whether the received address was a general call address (0x00). If the received address was a general call address, then the general call must be handled by user software.
- When 10-bit addressing is used to address a slave, the slave see an interrupt following the first byte of the extended address. User software must ensure that for this interrupt, the contents of IICD are ignored and not treated as a valid data transfer.



## NOTES:

- If general call siicaen is enabled, a check must be done to determine whether the received address was a general call address (0x00) or SMBus device default address. If the received address was one of them, then it must be handled by user software.
- Flow chart1 means Figure 17-11 Typical IIC Interrupt Routine.
- Delay about 1-2bit scl cycle waiting data register updated then clear IICIF

Figure 17-11. Typical IIC Interrupt Routine

# Chapter 18

## Modulo Timer (MTIM)

### 18.1 Introduction

The MTIM is a simple 8-bit timer with several software selectable clock sources and a programmable interrupt.

#### NOTE

- For MCUs that have more than one MTIM, the MTIMs are collectively called MTIMx. For example, MTIMx for an MCU with two MTIMs refers to MTIM1 and MTIM2. For MCUs that have exactly one MTIM, it is always referred to as MTIM1.
- Use pin mux control registers from [Section 2.3, “Pin Mux Controls”](#) to assign MTIM signals to the MCF51CN128 package pins.
- Most pin functions default to GPIO and must be software configured before using MTIM.
- The MCF51CN128 series MCUs do not include stop1 mode. Ignore all reference to stop1.

## 18.1.1 Features

Timer system features include:

- 8-bit up-counter
  - Free-running or 8-bit modulo limit
  - Software controllable interrupt on overflow
  - Counter reset bit (TRST)
  - Counter stop bit (TSTP)
- Four software selectable clock sources for input to prescaler:
  - System bus clock — rising edge
  - Fixed frequency clock (XCLK) — rising edge
  - External clock source on the TCLK pin — rising edge
  - External clock source on the TCLK pin — falling edge
- Nine selectable clock prescale values:
  - Clock source divide by 1, 2, 4, 8, 16, 32, 64, 128, or 256

## 18.1.2 Modes of Operation

This section defines the MTIM's operation in stop, wait, and background debug modes.

### 18.1.2.1 MTIM in Wait Mode

The MTIM continues to run in wait mode if enabled before executing the WAIT instruction. Therefore, the MTIM can be used to bring the MCU out of wait mode if the timer overflow interrupt is enabled. For lowest possible current consumption, the MTIM must be stopped by software if not needed as an interrupt source during wait mode.

### 18.1.2.2 MTIM in Stop Modes

The MTIM is disabled in all stop modes, regardless of the settings before executing the STOP instruction. Therefore, the MTIM cannot be used as a wake-up source from stop modes.

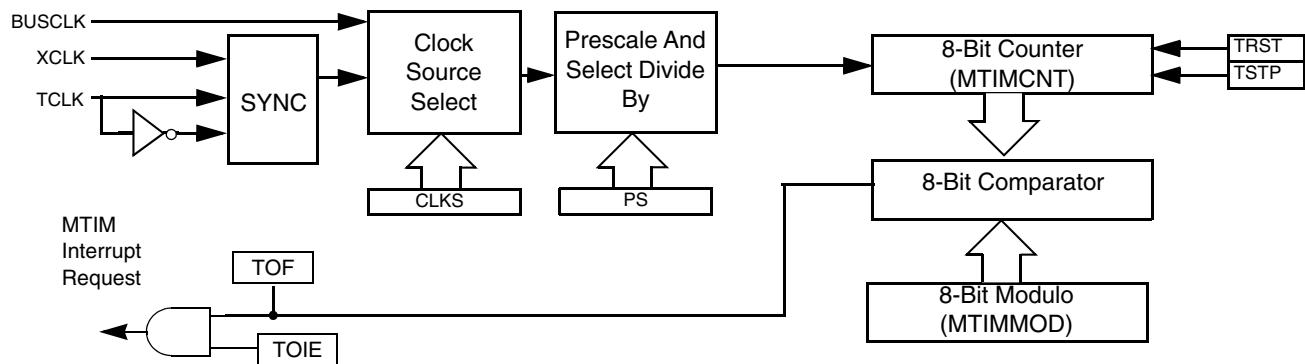
Waking from stop1 and stop2 modes, the MTIM is put into its reset state. If stop3 is exited with a reset, the MTIM is put into its reset state. If stop3 is exited with an interrupt, the MTIM continues from the state it was in when stop3 was entered. If the counter was active upon entering stop3, the count resumes from the current value.

### 18.1.2.3 MTIM in Active Background Mode

The MTIM suspends all counting until the microcontroller returns to normal user operating mode. Counting resumes from the suspended value as long as an MTIM reset did not occur (TRST written to a 1 or MTIMMOD written).

### 18.1.3 Block Diagram

The block diagram for the modulo timer module is shown [Figure 18-1](#).



**Figure 18-1. Modulo Timer (MTIM) Block Diagram**

### 18.2 External Signal Description

The MTIM includes one external signal, TCLK, used to input an external clock when selected as the MTIM clock source. The signal properties of TCLK are shown in [Table 18-1](#).

**Table 18-1. External Signal Description**

Signal	Function	I/O
TCLK	External clock source input into MTIM	I

The TCLK input must be synchronized by the bus clock. Also, variations in duty cycle and clock jitter must be accommodated. Therefore, the TCLK signal must be limited to one-fourth of the bus frequency.

The TCLK pin can be muxed with a general-purpose port pin. See [Chapter 2, “Pins and Connections”](#) for the pin location and priority of this function.

## 18.3 Register Definition

[Figure 18-2](#) is a summary of MTIM registers.

Name	7	6	5	4	3	2	1	0					
MTIMSC	R	TOF	TOIE	0	TSTP	0	0	0					
	W			TRST									
MTIMCLK	R	0	0	CLKS	PS								
	W												
MTIMCNT	R	COUNT											
	W												
MTIMMOD	R	MOD											
	W												

**Figure 18-2. MTIM Register Summary**

Each MTIM includes four registers:

- An 8-bit status and control register
- An 8-bit clock configuration register
- An 8-bit counter register
- An 8-bit modulo register

Refer to the direct-page register summary in [Chapter 4, “Memory”](#) for the absolute address assignments for all MTIM registers. This section refers to registers and control bits only by their names and relative address offsets.

Some MCUs may have more than one MTIM, so register names include placeholder characters to identify which MTIM is being referenced.

### 18.3.1 MTIM Status and Control Register (MTIMSC)

MTIMSC contains the overflow status flag and control bits that are used to configure the interrupt enable, reset the counter, and stop the counter.

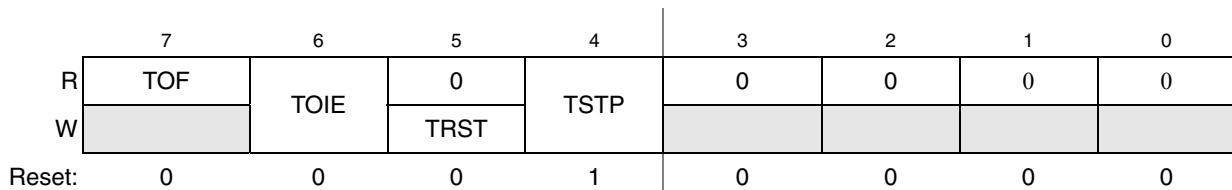


Figure 18-3. MTIM Status and Control Register

Table 18-2. MTIMSC Field Descriptions

Field	Description
7 TOF	<b>MTIM Overflow Flag</b> — This read-only bit is set when the MTIM counter register overflows to 0x00 after reaching the value in the MTIM modulo register. Clear TOF by reading the MTIMSC register while TOF is set, then writing a 0 to TOF. TOF is also cleared when TRST is written to a 1 or when any value is written to the MTIMMOD register. 0 MTIM counter has not reached the overflow value in the MTIM modulo register. 1 MTIM counter has reached the overflow value in the MTIM modulo register.
6 TOIE	<b>MTIM Overflow Interrupt Enable</b> — This read/write bit enables MTIM overflow interrupts. If TOIE is set, an interrupt is generated when TOF = 1. Reset clears TOIE. Do not set TOIE if TOF = 1. Clear TOF first, then set TOIE. 0 TOF interrupts are disabled. Use software polling. 1 TOF interrupts are enabled.
5 TRST	<b>MTIM Counter Reset</b> — When a 1 is written to this write-only bit, the MTIM counter register resets to 0x00 and TOF is cleared. Reading this bit always returns 0. 0 No effect. MTIM counter remains at current state. 1 MTIM counter is reset to 0x00.
4 TSTP	<b>MTIM Counter Stop</b> — When set, this read/write bit stops the MTIM counter at its current value. Counting resumes from the current value when TSTP is cleared. Reset sets TSTP to prevent the MTIM from counting. 0 MTIM counter is active. 1 MTIM counter is stopped.
3:0	Unused register bits, always read 0.

### 18.3.2 MTIM Clock Configuration Register (MTIMCLK)

MTIMCLK contains the clock select bits (CLKS) and the prescaler select bits (PS).

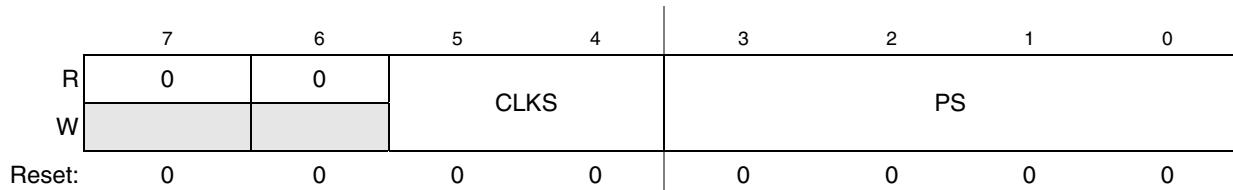


Figure 18-4. MTIM Clock Configuration Register

Table 18-3. MTIMCLK Field Descriptions

Field	Description
7:6	Unused register bits, always read 0.
5:4 CLKS	<b>Clock Source Select</b> — These two read/write bits select one of four different clock sources as the input to the MTIM prescaler. Changing the clock source while the counter is active does not clear the counter. The count continues with the new clock source. Reset clears CLKS to 000. 00 Encoding 0. Bus clock (BUSCLK) 01 Encoding 1. Fixed-frequency clock (XCLK) 10 Encoding 3. External source (TCLK pin), falling edge 11 Encoding 4. External source (TCLK pin), rising edge All other encodings default to the bus clock (BUSCLK).
3:0 PS	<b>Clock Source Prescaler</b> — These four read/write bits select one of nine outputs from the 8-bit prescaler. Changing the prescaler value while the counter is active does not clear the counter. The count continues with the new prescaler value. Reset clears PS to 0000. 0000 Encoding 0. MTIM clock source $\div 1$ 0001 Encoding 1. MTIM clock source $\div 2$ 0010 Encoding 2. MTIM clock source $\div 4$ 0011 Encoding 3. MTIM clock source $\div 8$ 0100 Encoding 4. MTIM clock source $\div 16$ 0101 Encoding 5. MTIM clock source $\div 32$ 0110 Encoding 6. MTIM clock source $\div 64$ 0111 Encoding 7. MTIM clock source $\div 128$ 1000 Encoding 8. MTIM clock source $\div 256$ All other encodings default to MTIM clock source $\div 256$ .

### 18.3.3 MTIM Counter Register (MTIMCNT)

MTIMCNT is the read-only value of the current MTIM count of the 8-bit counter.

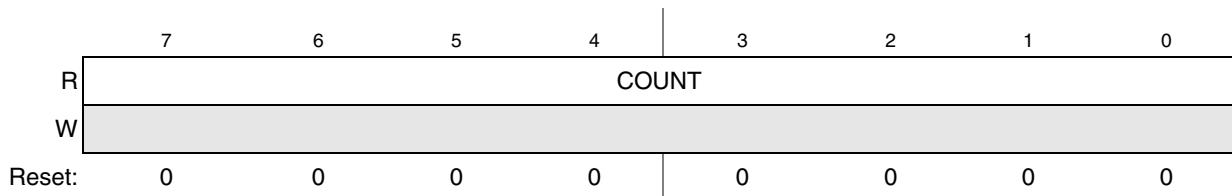


Figure 18-5. MTIM Counter Register

Table 18-4. MTIM Field Descriptions

Field	Description
7:0 COUNT	<b>MTIM Count</b> — These eight read-only bits contain the current value of the 8-bit counter. Writes have no effect to this register. Reset clears the count to 0x00.

### 18.3.4 MTIM Modulo Register (MTIMMOD)

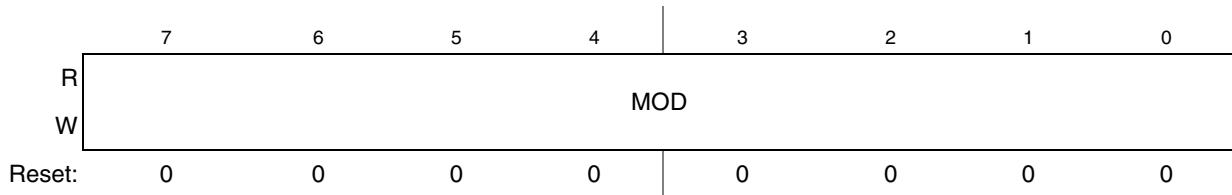


Figure 18-6. MTIM Modulo Register

Table 18-5. MTIM Modulo Register

Field	Description
7:0 MOD	<b>MTIM Modulo</b> — These eight read/write bits contain the modulo value used to reset the count and set TOF. A value of 0x00 puts the MTIM in free-running mode. Writing to MTIMMOD resets the COUNT to 0x00 and clears TOF. Reset sets the modulo to 0x00.

## 18.4 Functional Description

The MTIM is composed of a main 8-bit up-counter with an 8-bit modulo register, a clock source selector, and a prescaler block with nine selectable values. The module also contains software selectable interrupt logic.

The MTIM counter (MTIMCNT) has three modes of operation: stopped, free-running, and modulo. Out of reset, the counter is stopped. If the counter is started without writing a new value to the modulo register, the counter is in free-running mode. The counter is in modulo mode when a value other than 0x00 is in the modulo register while the counter is running.

After any MCU reset, the counter is stopped and reset to 0x00, and the modulus is set to 0x00. The bus clock is selected as the default clock source and the prescale value is divide by one. To start the MTIM in free-running mode, simply write to the MTIM status and control register (MTIMSC) and clear the MTIM stop bit (TSTP).

Four clock sources are software selectable: the internal bus clock, the fixed frequency clock (XCLK), and an external clock on the TCLK pin, selectable as incrementing on either rising or falling edges. The MTIM clock select bits (CLKS1:CLKS0) in MTIMSC are used to select the desired clock source. If the counter is active (TSTP = 0) when a new clock source is selected, the counter continues counting from the previous value using the new clock source.

Nine prescale values are software selectable: clock source divided by 1, 2, 4, 8, 16, 32, 64, 128, or 256. The prescaler select bits (PS[3:0]) in MTIMSC select the desired prescale value. If the counter is active (TSTP = 0) when a new prescaler value is selected, the counter continues counting from the previous value using the new prescaler value.

The MTIM modulo register (MTIMMOD) allows the overflow compare value to be set to any value from 0x01 to 0xFF. Reset clears the modulo value to 0x00 that results in a free running counter.

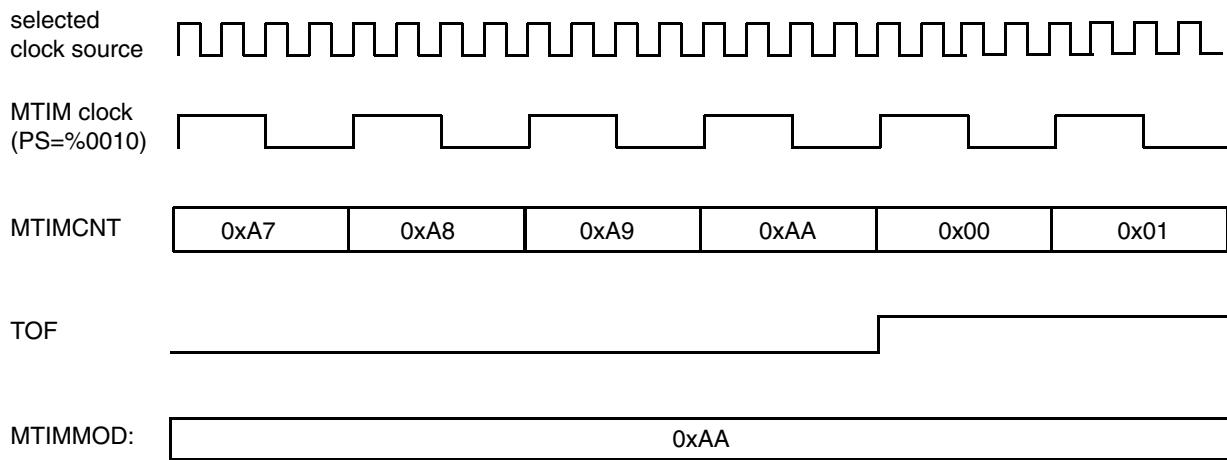
When the counter is active (TSTP = 0), the counter increments at the selected rate until the count matches the modulo value. When these values match, the counter overflows to 0x00 and continues counting. The MTIM overflow flag (TOF) is set whenever the counter overflows. The flag sets on the transition from the modulo value to 0x00. Writing to MTIMMOD while the counter is active resets the counter to 0x00 and clears TOF.

Clearing TOF is a two-step process. The first step is to read the MTIMSC register while TOF is set. The second step is to write a 0 to TOF. If another overflow occurs between the first and second steps, the clearing process is reset and TOF remains set after the second step is performed. This prevents the second occurrence from being missed. TOF is also cleared when a 1 is written to TRST or when any value is written to the MTIMMOD register.

The MTIM allows for an optional interrupt to be generated whenever TOF is set. To enable the MTIM overflow interrupt, set the MTIM overflow interrupt enable bit (TOIE) in MTIMSC. TOIE must not be written to a 1 while TOF = 1. Instead, TOF must be cleared first, then the TOIE can be set to 1.

### 18.4.1 MTIM Operation Example

This section shows an example of the MTIM operation as the counter reaches a matching value from the modulo register.



**Figure 18-7. MTIM counter overflow example**

In the example of [Figure 18-7](#), the selected clock source could be any of the five possible choices. The prescaler is set to PS = %0010 or divide-by-4. The modulo value in the MTIMMOD register is set to 0xAA. When the counter, MTIMCNT, reaches the modulo value of 0xAA, the counter overflows to 0x00 and continues counting. The timer overflow flag, TOF, sets when the counter value changes from 0xAA to 0x00. An MTIM overflow interrupt is generated when TOF is set, if TOIE = 1.

# Chapter 19

## Timer/PWM Module (TPM)

### 19.1 Introduction

The TPM is a one-to-eight-channel timer system that supports traditional input capture, output compare, or edge-aligned PWM on each channel. A control bit configures the TPM so all channels are used for center-aligned PWM functions. Timing functions are based on a 16-bit counter with prescaler and modulo features to control frequency and range (period between overflows) of the time reference. This timing system is ideally suited for a wide range of control applications, and the center-aligned PWM capability extends the field of application to motor control in small appliances.

#### NOTE

- Use pin mux control registers from [Section 2.3, “Pin Mux Controls”](#) to assign TPM signals to the MCF51CN128 package pins.
- Most pin functions default to GPIO and must be software configured before using TPM.

### 19.1.1 Features

The TPM includes these distinctive features:

- One to eight channels:
  - Each channel is input capture, output compare, or edge-aligned PWM
  - Rising-edge, falling-edge, or any-edge input capture trigger
  - Set, clear, or toggle output compare action
  - Selectable polarity on PWM outputs
- Module is configured for buffered, center-aligned pulse-width-modulation (CPWM) on all channels
- Timer clock source selectable as bus clock, fixed frequency clock, or an external clock
  - Prescale taps for divide-by 1, 2, 4, 8, 16, 32, 64, or 128 used for any clock input selection
  - Fixed frequency clock is an additional clock input to allow the selection of an on chip clock source other than bus clock
  - Selecting external clock connects TPM clock to a chip level input pin therefore allowing to synchronize the TPM counter with an off chip clock source
- 16-bit free-running or modulus count with up/down selection
- One interrupt per channel and one interrupt for TPM counter overflow

### 19.1.2 Modes of Operation

In general, TPM channels are independently configured to operate in input capture, output compare, or edge-aligned PWM modes. A control bit allows the whole TPM (all channels) to switch to center-aligned PWM mode. When center-aligned PWM mode is selected, input capture, output compare, and edge-aligned PWM functions are not available on any channels of this TPM module.

When the MCU is in active BDM background or BDM foreground mode, the TPM temporarily suspends all counting until the MCU returns to normal user operating mode. During stop mode, all TPM input clocks are stopped, so the TPM is effectively disabled until clocks resume. During wait mode, the TPM continues to operate normally. If the TPM does not need to produce a real time reference or provide the interrupt sources needed to wake the MCU from wait mode, the power can then be saved by disabling TPM functions before entering wait mode.

- Input capture mode
 

When a selected edge event occurs on the associated MCU pin, the current value of the 16-bit timer counter is captured into the channel value register and an interrupt flag bit is set. Rising edges, falling edges, any edge, or no edge (disable channel) are selected as the active edge that triggers the input capture.
- Output compare mode
 

When the value in the timer counter register matches the channel value register, an interrupt flag bit is set, and a selected output action is forced on the associated MCU pin. The output compare action is selected to force the pin to zero, force the pin to one, toggle the pin, or ignore the pin (used for software timing functions).
- Edge-aligned PWM mode

The value of a 16-bit modulo register plus 1 sets the period of the PWM output signal. The channel value register sets the duty cycle of the PWM output signal. You can also choose the polarity of the PWM output signal. Interrupts are available at the end of the period and at the duty-cycle transition point. This type of PWM signal is called edge-aligned because the leading edges of all PWM signals are aligned with the beginning of the period that is same for all channels within a TPM.

- Center-aligned PWM mode

Twice the value of a 16-bit modulo register sets the period of the PWM output, and the channel-value register sets the half-duty-cycle duration. The timer counter counts up until it reaches the modulo value and then counts down until it reaches zero. As the count matches the channel value register while counting down, the PWM output becomes active. When the count matches the channel value register while counting up, the PWM output becomes inactive. This type of PWM signal is called center-aligned because the centers of the active duty cycle periods for all channels are aligned with a count value of zero. This type of PWM is required for types of motors used in small appliances.

This is a high-level description only. Detailed descriptions of operating modes are in later sections.

### 19.1.3 Block Diagram

The TPM uses one input/output (I/O) pin per channel, TPMxCH<sub>n</sub> (timer channel n) where n is the channel number (1–8). The TPM shares its I/O pins with general purpose I/O port pins (refer to I/O pin descriptions in full-chip specification for the specific chip implementation).

[Figure 19-1](#) shows the TPM structure. The central component of the TPM is the 16-bit counter that can operate as a free-running counter or a modulo up/down counter. The TPM counter (when operating in normal up-counting mode) provides the timing reference for the input capture, output compare, and edge-aligned PWM functions. The timer counter modulo registers, TPMxMODH:TPMxMODL, control the modulo value of the counter (the values 0x0000 or 0xFFFF effectively make the counter free running). Software can read the counter value at any time without affecting the counting sequence. Any write to either half of the TPMxCNT counter resets the counter, regardless of the data value written.

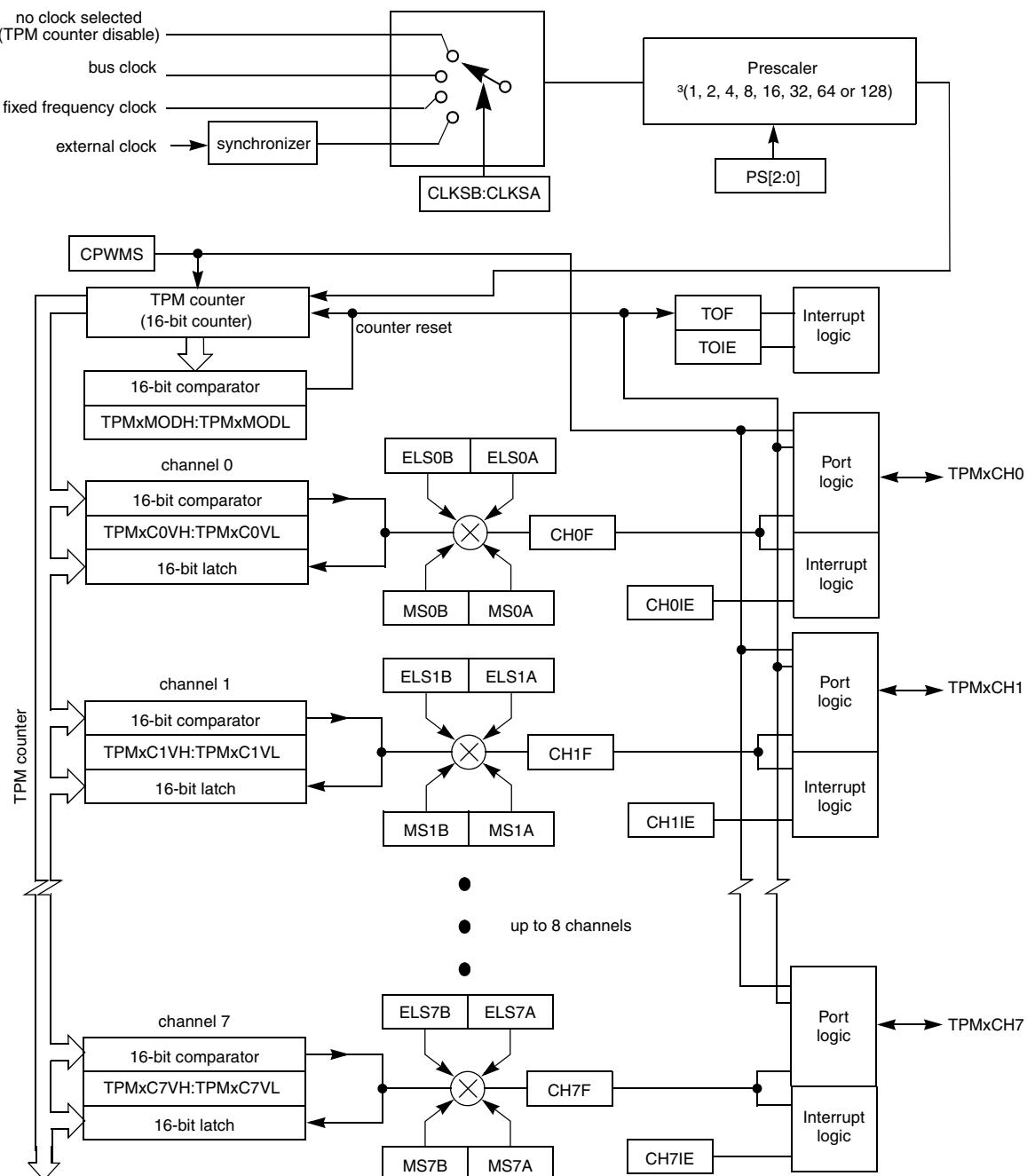


Figure 19-1. TPM Block Diagram

The TPM channels are programmable independently as input capture, output compare, or edge-aligned PWM channels. Alternately, the TPM can be configured to produce CPWM outputs on all channels. When the TPM is configured for CPWMS (the counter operates as an up/down counter) input capture, output compare, and EPWM functions are not practical.

## 19.2 Signal Description

**Table 19-1** shows the user-accessible signals for the TPM. The number of channels are varied from one to eight. When an external clock is included, it can be shared with the same pin as any TPM channel; however, it could be connected to a separate input pin. Refer to the I/O pin descriptions in full-chip specification for the specific chip implementation.

**Table 19-1. Signal Properties**

Name	Function
EXTCLK <sup>1</sup>	External clock source that is selected to drive the TPM counter.
TPMxCHn <sup>2</sup>	I/O pin associated with TPM channel n.

<sup>1</sup> The external clock pin can be shared with any channel pin. However, depending upon full-chip implementation, this signal could be connected to a separate external pin.

<sup>2</sup> n = channel number (1–8)

### 19.2.1 Detailed Signal Descriptions

#### 19.2.1.1 EXTCLK — External Clock Source

The external clock signal can share the same pin as a channel pin, however the channel pin can not be used for channel I/O function when external clock is selected. If this pin is used as an external clock (CLKSB:CLKSA = 1:1), the channel can still be configured to output compare mode therefore allowing its use as a timer (ELSnB:ELSnA = 0:0).

For proper TPM operation, the external clock frequency must not exceed one-fourth of the bus clock frequency.

#### 19.2.1.2 TPMxCHn — TPM Channel n I/O Pins

The TPM channel does not control the I/O pin when ELSnB:ELSnA or CLKSB:CLKSA are cleared so it normally reverts to general purpose I/O control. When CPWMS is set and ELSnB:ELSnA are not cleared, all TPM channels are configured for center-aligned PWM and the TPMxCHn pins are all controlled by TPM. When CPWMS is cleared, the MSnB:MSnA control bits determine whether the channel is configured for input capture, output compare, or edge-aligned PWM.

When a channel is configured for input capture (CPWMS = 0, MSnB:MSnA = 0:0, and ELSnB:ELSnA ≠ 0:0), the TPMxCHn pin is forced to act as an edge-sensitive input to the TPM. ELSnB:ELSnA control bits determine what polarity edge or edges trigger input capture events. The channel input signal is synchronized on the bus clock. This implies the minimum pulse width—that can

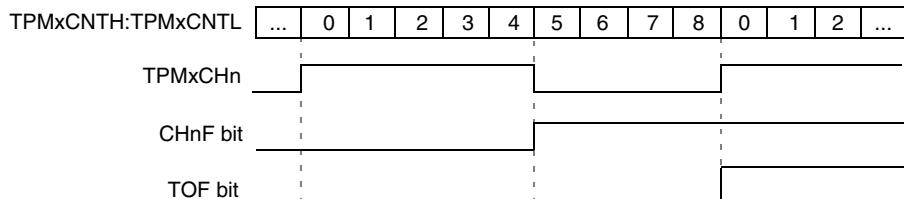
be reliably detected—on an input capture pin is four bus clock periods (with ideal clock pulses as near as two bus clocks can be detected).

When a channel is configured for output compare (CPWMS = 0, MSnB:MSnA = 0:1, and ELSnB:ELSnA  $\neq$  0:0), the TPMxCHn pin is an output controlled by the TPM. The ELSnB:ELSnA bits determine whether the TPMxCHn pin is toggled, cleared, or set each time the 16-bit channel value register matches the TPM counter.

When the output compare toggle mode is initially selected, the previous value on the pin is driven out until the next output compare event, the pin is then toggled.

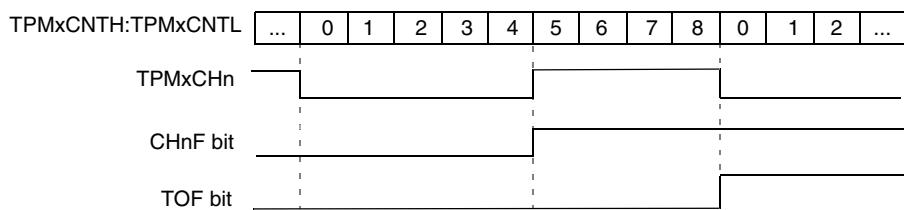
When a channel is configured for edge-aligned PWM (CPWMS = 0, MSnB = 1, and ELSnB:ELSnA  $\neq$  0:0), the TPMxCHn pin is an output controlled by the TPM, and ELSnB:ELSnA bits control the polarity of the PWM output signal. When ELSnB is set and ELSnA is cleared, the TPMxCHn pin is forced high at the start of each new period (TPMxCNT=0x0000), and it is forced low when the channel value register matches the TPM counter. When ELSnA is set, the TPMxCHn pin is forced low at the start of each new period (TPMxCNT=0x0000), and it is forced high when the channel value register matches the TPM counter.

TPMxMODH:TPMxMODL = 0x0008  
TPMxCnVH:TPMxCnVL = 0x0005



**Figure 19-2. High-True Pulse of an Edge-Aligned PWM**

TPMxMODH:TPMxMODL = 0x0008  
TPMxCnVH:TPMxCnVL = 0x0005



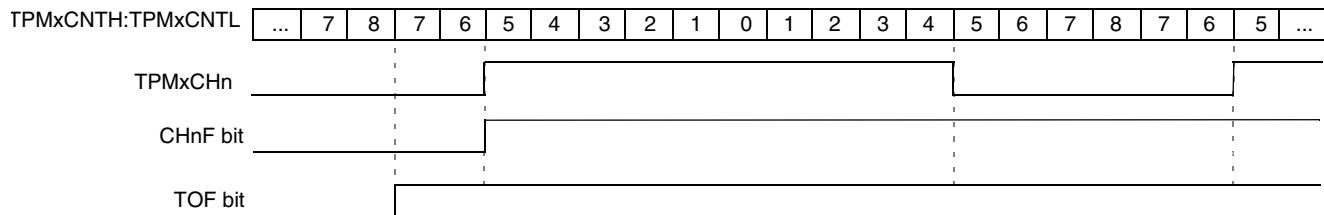
**Figure 19-3. Low-True Pulse of an Edge-Aligned PWM**

When the TPM is configured for center-aligned PWM (CPWMS = 1 and ELSnB:ELSnA  $\neq$  0:0), the TPMxCHn pins are outputs controlled by the TPM, and ELSnB:ELSnA bits control the polarity of the PWM output signal. If ELSnB is set and ELSnA is cleared, the corresponding TPMxCHn pin is cleared when the TPM counter is counting up, and the channel value register matches the TPM counter; and it is

## Timer/PWM Module (TPM)

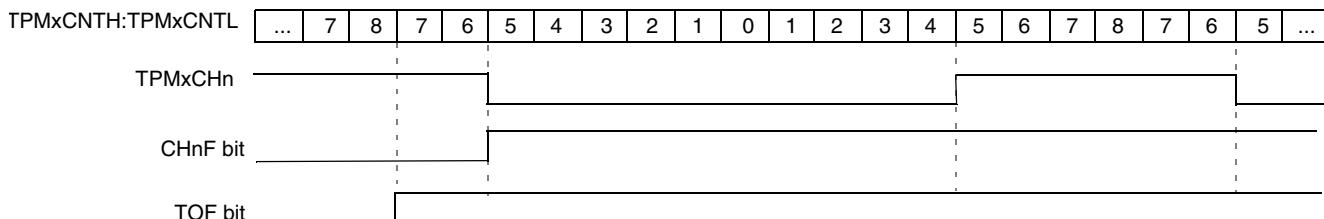
set when the TPM counter is counting down, and the channel value register matches the TPM counter. If ELSnA is set, the corresponding TPMxCHn pin is set when the TPM counter is counting up and the channel value register matches the TPM counter; and it is cleared when the TPM counter is counting down and the channel value register matches the TPM counter.

TPMxMODH:TPMxMODL = 0x0008  
TPMxCnVH:TPMxCnVL = 0x0005



**Figure 19-4. High-True Pulse of a Center-Aligned PWM**

TPMxMODH:TPMxMODL = 0x0008  
TPMxCnVH:TPMxCnVL = 0x0005



**Figure 19-5. Low-True Pulse of a Center-Aligned PWM**

## 19.3 Register Definition

### 19.3.1 TPM Status and Control Register (TPMxSC)

TPMxSC contains the overflow status flag and control bits used to configure the interrupt enable, TPM configuration, clock source, and prescale factor. These controls relate to all channels within this timer module.

	7	6	5	4	3	2	1	0
R	TOF	TOIE	CPWMS	CLKSB	CLKSA	PS2	PS1	PS0
W	0	0	0	0	0	0	0	0
Reset	0	0	0	0	0	0	0	0

Figure 19-6. TPM Status and Control Register (TPMxSC)

Table 19-2. TPMxSC Field Descriptions

Field	Description
7 TOF	Timer overflow flag. This read/write flag is set when the TPM counter resets to 0x0000 after reaching the modulo value programmed in the TPM counter modulo registers. Clear TOF by reading the TPM status and control register when TOF is set and then writing a logic 0 to TOF. If another TPM overflow occurs before the clearing sequence is completed, the sequence is reset so TOF remains set after the clear sequence was completed for the earlier TOF. This is done so a TOF interrupt request cannot be lost during the clearing sequence for a previous TOF. Reset clears TOF. Writing a logic 1 to TOF has no effect. 0 TPM counter has not reached modulo value or overflow. 1 TPM counter has overflowed.
6 TOIE	Timer overflow interrupt enable. This read/write bit enables TPM overflow interrupts. If TOIE is set, an interrupt is generated when TOF equals one. Reset clears TOIE. 0 TOF interrupts inhibited (use for software polling). 1 TOF interrupts enabled.
5 CPWMS	Center-aligned PWM select. This read/write bit selects CPWM operating mode. By default, the TPM operates in up-counting mode for input capture, output compare, and edge-aligned PWM functions. Setting CPWMS reconfigures the TPM to operate in up/down counting mode for CPWM functions. Reset clears CPWMS. 0 All channels operate as input capture, output compare, or edge-aligned PWM mode as selected by the MSnB:MSnA control bits in each channel's status and control register. 1 All channels operate in center-aligned PWM mode.
4–3 CLKS[B:A]	Clock source selection bits. As shown in Table 19-3, this 2-bit field is used to disable the TPM counter or select one of three clock sources to TPM counter and counter prescaler.
2–0 PS[2:0]	Prescale factor select. This 3-bit field selects one of eight division factors for the TPM clock as shown in Table 19-4. This prescaler is located after any clock synchronization or clock selection so it affects the clock selected to drive the TPM counter. The new prescale factor affects the selected clock on the next bus clock cycle after the new value is updated into the register bits.

Table 19-3. TPM Clock Selection

CLKSB:CLKSA	TPM Clock to Prescaler Input
00	No clock selected (TPM counter disable)
01	Bus clock

**Table 19-3. TPM Clock Selection (continued)**

<b>CLKSB:CLKSA</b>	<b>TPM Clock to Prescaler Input</b>
10	Fixed frequency clock
11	External clock

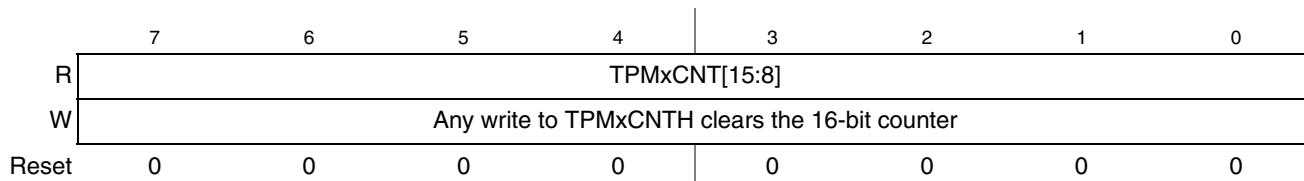
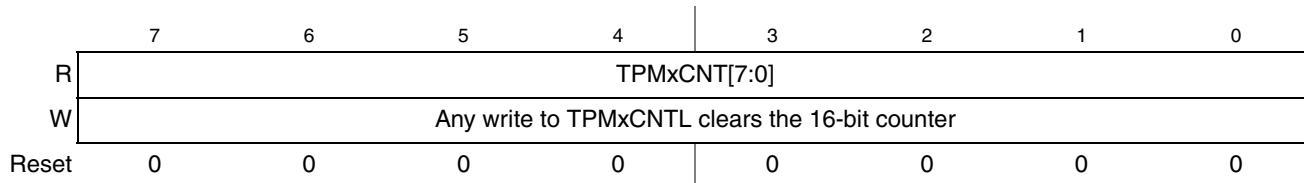
**Table 19-4. Prescale Factor Selection**

<b>PS[2:0]</b>	<b>TPM Clock Divided-by</b>
000	1
001	2
010	4
011	8
100	16
101	32
110	64
111	128

### 19.3.2 TPM-Counter Registers (TPMxCNTH:TPMxCNTL)

The two read-only TPM counter registers contain the high and low bytes of the value in the TPM counter. Reading either byte (TPMxCNTH or TPMxCNTL) latches the contents of both bytes into a buffer where they remain latched until the other half is read. This allows coherent 16-bit reads in big-endian or little-endian order that makes this more friendly to various compiler implementations. The coherency mechanism is automatically restarted by an MCU reset or any write to the timer status/control register (TPMxSC).

Reset clears the TPM counter registers. Writing any value to TPMxCNTH or TPMxCNTL also clears the TPM counter (TPMxCNTH:TPMxCNTL) and resets the coherency mechanism, regardless of the data involved in the write.

**Figure 19-7. TPM Counter Register High (TPMxCNTH)****Figure 19-8. TPM Counter Register Low (TPMxCNTL)**

When BDM is active, the timer counter is frozen (this is the value you read). The coherency mechanism is frozen so the buffer latches remain in the state they were in when the BDM became active, even if one or both counter halves are read while BDM is active. This assures that if you were in the middle of reading a 16-bit register when BDM became active, it reads the appropriate value from the other half of the 16-bit value after returning to normal execution.

In BDM mode, writing any value to TPMxSC, TPMxCNTH, or TPMxCNTL registers resets the read coherency mechanism of the TPMxCNTH:TPMxCNTL registers, regardless of the data involved in the write.

### 19.3.3 TPM Counter Modulo Registers (TPMxMODH:TPMxMODL)

The read/write TPM modulo registers contain the modulo value for the TPM counter. After the TPM counter reaches the modulo value, the TPM counter resumes counting from 0x0000 at the next clock, and the overflow flag (TOF) becomes set. Writing to TPMxMODH or TPMxMODL inhibits the TOF bit and overflow interrupts until the other byte is written. Reset sets the TPM counter modulo registers to 0x0000 that results in a free running timer counter (modulo disabled).

Writes to any of the registers TPMxMODH and TPMxMODL actually writes to buffer registers and the registers are updated with the value of their write buffer according to the value of CLKSB:CLKSA bits:

- If CLKSB and CLKSA are cleared, the registers are updated when the second byte is written
- If CLKSB and CLKSA are not cleared, the registers are updated after both bytes were written, and the TPM counter changes from (TPMxMODH:TPMxMODL – 1) to (TPMxMODH:TPMxMODL). If the TPM counter is a free-running counter, the update is made when the TPM counter changes from 0xFFFF to 0xFFFF

The latching mechanism is manually reset by writing to the TPMxSC address (whether BDM is active or not).

When BDM is active, the coherency mechanism is frozen (unless reset by writing to TPMxSC register) so the buffer latches remain in the state they were in when the BDM became active, even if one or both halves of the modulo register are written while BDM is active. Any write to the modulo registers bypasses the buffer latches and directly writes to the modulo register while BDM is active.

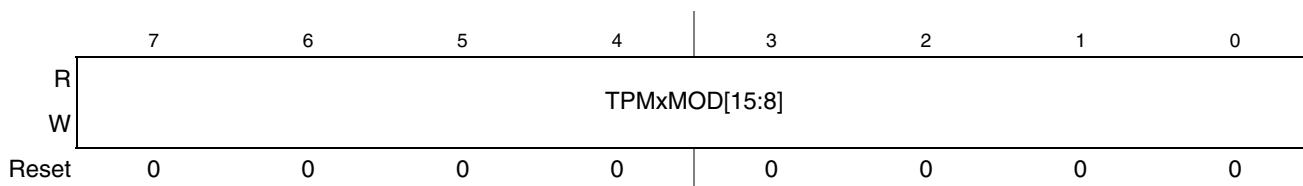


Figure 19-9. TPM Counter Modulo Register High (TPMxMODH)

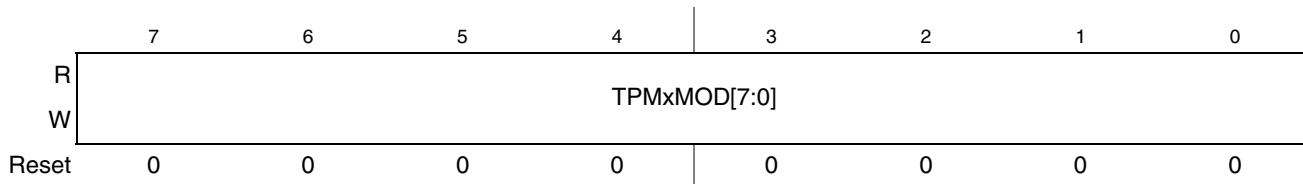


Figure 19-10. TPM Counter Modulo Register Low (TPMxMODL)

Reset the TPM counter before writing to the TPM modulo registers to avoid confusion about when the first counter overflow occurs.

### 19.3.4 TPM Channel n Status and Control Register (TPMxCnSC)

TPMxCnSC contains the channel-interrupt-status flag and control bits that configure the interrupt enable, channel configuration, and pin function.

	7	6	5	4	3	2	1	0
R	CHnF	CHnIE	MSnB	MSnA	ELSnB	ELSnA	0	0
W	0							
Reset	0	0	0	0	0	0	0	0
	[Gray Box]	= Unimplemented or Reserved						

Figure 19-11. TPM Channel n Status and Control Register (TPMxCnSC)

Table 19-5. TPMxCnSC Field Descriptions

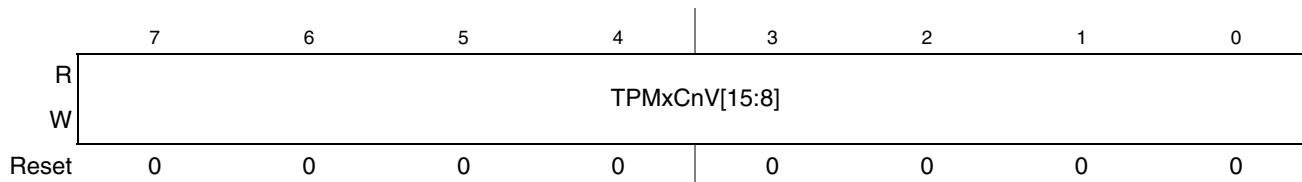
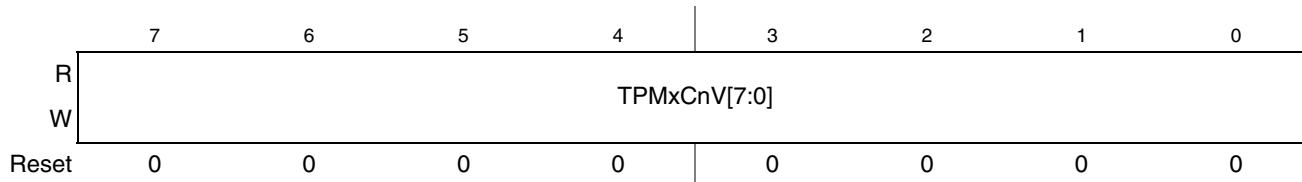
Field	Description
7 CHnF	Channel n flag. When channel n is an input capture channel, this read/write bit is set when an active edge occurs on the channel n input. When channel n is an output compare or edge-aligned/center-aligned PWM channel, CHnF is set when the value in the TPM counter registers matches the value in the TPM channel n value registers. When channel n is an edge-aligned/center-aligned PWM channel and the duty cycle is set to 0% or 100%, CHnF is not set even when the value in the TPM counter registers matches the value in the TPM channel n value registers. A corresponding interrupt is requested when this bit is set and channel n interrupt is enabled (CHnIE = 1). Clear CHnF by reading TPMxCnSC while this bit is set and then writing a logic 0 to it. If another interrupt request occurs before the clearing sequence is completed CHnF remains set. This is done so a CHnF interrupt request is not lost due to clearing a previous CHnF. Reset clears this bit. Writing a logic 1 to CHnF has no effect. 0 No input capture or output compare event occurred on channel n. 1 Input capture or output compare event on channel n.
6 CHnIE	Channel n interrupt enable. This read/write bit enables interrupts from channel n. Reset clears this bit. 0 Channel n interrupt requests disabled (use for software polling). 1 Channel n interrupt requests enabled.
5 MSnB	Mode select B for TPM channel n. When CPWMS is cleared, setting the MSnB bit configures TPM channel n for edge-aligned PWM mode. Refer to the summary of channel mode and setup controls in <a href="#">Table 19-6</a> .
4 MSnA	Mode select A for TPM channel n. When CPWMS and MSnB are cleared, the MSnA bit configures TPM channel n for input capture mode or output compare mode. Refer to <a href="#">Table 19-6</a> for a summary of channel mode and setup controls. <b>Note:</b> If the associated port pin is not stable for at least two bus clock cycles before changing to input capture mode, it is possible to get an unexpected indication of an edge trigger.
3-2 ELSnB ELSnA	Edge/level select bits. Depending upon the operating mode for the timer channel as set by CPWMS:MSnB:MSnA and shown in <a href="#">Table 19-6</a> , these bits select the polarity of the input edge that triggers an input capture event, select the level that is driven in response to an output compare match, or select the polarity of the PWM output. If ELSnB and ELSnA bits are cleared, the channel pin is not controlled by TPM. This configuration can be used by software compare only, because it does not require the use of a pin for the channel.

**Table 19-6. Mode, Edge, and Level Selection**

<b>CPWMS</b>	<b>MSnB:MSnA</b>	<b>ELSnB:ELSnA</b>	<b>Mode</b>	<b>Configuration</b>	
X	XX	00	Pin is not controlled by TPM. It is reverted to general purpose I/O or other peripheral control		
0	00	01	Input capture	Capture on rising edge only	
		10		Capture on falling edge only	
		11		Capture on rising or falling edge	
	01	00	Output compare	Software compare only	
		01		Toggle output on channel match	
		10		Clear output on channel match	
		11		Set output on channel match	
	1X	10	Edge-aligned PWM	High-true pulses (clear output on channel match)	
		X1		Low-true pulses (set output on channel match)	
1	XX	10	Center-aligned PWM	High-true pulses (clear output on channel match when TPM counter is counting up)	
		X1		Low-true pulses (set output on channel match when TPM counter is counting up)	

### 19.3.5 TPM Channel Value Registers (TPMxCnVH:TPMxCnVL)

These read/write registers contain the captured TPM counter value of the input capture function or the output compare value for the output compare or PWM functions. The channel registers are cleared by reset.

**Figure 19-12. TPM Channel Value Register High (TPMxCnVH)****Figure 19-13. TPM Channel Value Register Low (TPMxCnVL)**

In input capture mode, reading either byte (TPMxCnVH or TPMxCnVL) latches the contents of both bytes into a buffer where they remain latched until the other half is read. This latching mechanism also resets (becomes unlatched) when the TPMxCnSC register is written (whether BDM mode is active or not). Any write to the channel registers is ignored during the input capture mode.

When BDM is active, the coherency mechanism is frozen (unless reset by writing to TPMxCnSC register) so the buffer latches remain in the state they were in when the BDM became active, even if one or both halves of the channel register are read while BDM is active. This assures that if you were in the middle of reading a 16-bit register when BDM became active, it reads the appropriate value from the other half of the 16-bit value after returning to normal execution. The value read from the TPMxCnVH and TPMxCnVL registers in BDM mode is the value of these registers and not the value of their read buffer.

In output compare or PWM modes, writing to either byte (TPMxCnVH or TPMxCnVL) latches the value into a buffer. After both bytes were written, they are transferred as a coherent 16-bit value into the timer-channel registers according to the value of CLKSB:CLKSA bits and the selected mode:

- If CLKSB and CLKSA are cleared, the registers are updated when the second byte is written.
- If CLKSB and CLKSA are not cleared and in output compare mode, the registers are updated after the second byte is written and on the next change of the TPM counter (end of the prescaler counting).
- If CLKSB and CLKSA are not cleared and in EPWM or CPWM modes, the registers are updated after both bytes were written, and the TPM counter changes from (TPMxMODH:TPMxMODL – 1) to (TPMxMODH:TPMxMODL). If the TPM counter is a free-running counter, the update is made when the TPM counter changes from 0xFFFF to 0xFFFF.

The latching mechanism is manually reset by writing to the TPMxCnSC register (whether BDM mode is active or not). This latching mechanism allows coherent 16-bit writes in either big-endian or little-endian order that is friendly to various compiler implementations.

When BDM is active, the coherency mechanism is frozen so the buffer latches remain in the state they were in when the BDM became active even if one or both halves of the channel register are written while BDM is active. Any write to the channel registers bypasses the buffer latches and directly write to the channel register while BDM is active. The values written to the channel register while BDM is active are used for PWM and output compare operation after normal execution resumes. Writes to the channel registers while BDM is active do not interfere with partial completion of a coherency sequence. After the coherency mechanism is fully exercised, the channel registers are updated using the buffered values (while BDM was not active).

## 19.4 Functional Description

All TPM functions are associated with a central 16-bit counter that allows flexible selection of the clock and prescale factor. There is also a 16-bit modulo register associated with this counter.

The CPWMS control bit chooses between center-aligned PWM operation for all channels in the TPM (CPWMS=1) or general purpose timing functions (CPWMS=0) where each channel can independently be configured to operate in input capture, output compare, or edge-aligned PWM mode. The CPWMS control bit is located in the TPM status and control register because it affects all channels within the TPM and influences the way the main counter operates. (In CPWM mode, the counter changes to an up/down mode rather than the up-counting mode used for general purpose timer functions.)

The following sections describe TPM counter and each of the timer operating modes (input capture, output compare, edge-aligned PWM, and center-aligned PWM). Because details of pin operation and interrupt

activity depend upon the operating mode, these topics are covered in the associated mode explanation sections.

## 19.4.1 Counter

All timer functions are based on the main 16-bit counter (TPMxCNTH:TPMxCNTL). This section discusses selection of the clock, end-of-count overflow, up-counting vs. up/down counting, and manual counter reset.

### 19.4.1.1 Counter Clock Source

The 2-bit field, CLKSB:CLKSA, in the timer status and control register (TPMxSC) disables the TPM counter or selects one of three clock sources to TPM counter ([Table 19-3](#)). After any MCU reset, CLKSB and CLKSA are cleared so no clock is selected and the TPM counter is disabled (TPM is in a very low power state). You can read or write these control bits at any time. Disabling the TPM counter by writing 00 to CLKSB:CLKSA bits, does not affect the values in the TPM counter or other registers.

The fixed frequency clock is an alternative clock source for the TPM counter that allows the selection of a clock other than the bus clock or external clock. This clock input is defined by chip integration. You can refer chip specific documentation for further information. Due to TPM hardware implementation limitations, the frequency of the fixed frequency clock must not exceed the bus clock frequency. The fixed frequency clock has no limitations for low frequency operation.

The external clock passes through a synchronizer clocked by the bus clock to assure that counter transitions are properly aligned to bus clock transitions. Therefore, in order to meet Nyquist criteria considering also jitter, the frequency of the external clock source must not exceed 1/4 of the bus clock frequency.

When the external clock source is shared with a TPM channel pin, this pin must not be used in input capture mode. However, this channel can be used in output compare mode with ELSnB:ELSnA = 0:0 for software timing functions. In this case, the channel output is disabled, but the channel match events continue to set the appropriate flag.

### 19.4.1.2 Counter Overflow and Modulo Reset

An interrupt flag and enable are associated with the 16-bit main counter. The flag (TOF) is a software-accessible indication that the timer counter has overflowed. The enable signal selects between software polling (TOIE = 0) where no interrupt is generated, or interrupt-driven operation (TOIE = 1) where the interrupt is generated whenever the TOF is set.

The conditions causing TOF to become set depend on whether the TPM is configured for center-aligned PWM (CPWMS = 1). If CPWMS is cleared and there is no modulus limit, the 16-bit timer counter counts from 0x0000 through 0xFFFF and overflows to 0x0000 on the next counting clock. TOF is set at the transition from 0xFFFF to 0x0000. When a modulus limit is set, TOF is set at the transition from the value set in the modulus register to 0x0000. When the TPM is in center-aligned PWM mode (CPWMS = 1), the TOF flag is set as the counter changes direction at the end of the count value set in the modulus register (at the transition from the value set in the modulus register to the next lower count value). This corresponds to the end of a PWM period (the 0x0000 count value corresponds to the center of a period).

### 19.4.1.3 Counting Modes

The main timer counter has two counting modes. When center-aligned PWM is selected (CPWMS = 1), the counter operates in up/down counting mode. Otherwise, the counter operates as a simple up counter. As an up counter, the timer counter counts from 0x0000 through its terminal count and continues with 0x0000. The terminal count is 0xFFFF or a modulus value in TPMxMODH:TPMxMODL.

When center-aligned PWM operation is specified, the counter counts up from 0x0000 through its terminal count and then down to 0x0000 where it changes back to up counting. The terminal count value and 0x0000 are normal length counts (one timer clock period long). In this mode, the timer overflow flag (TOF) is set at the end of the terminal-count period (as the count changes to the next lower count value).

### 19.4.1.4 Manual Counter Reset

The main timer counter can be manually reset at any time by writing any value to TPMxCNTH or TPMxCNTL. Resetting the counter in this manner also resets the coherency mechanism in case only half of the counter was read before resetting the count.

## 19.4.2 Channel Mode Selection

If CPWMS is cleared, MSnB and MSnA bits determine the basic mode of operation for the corresponding channel. Choices include input capture, output compare, and edge-aligned PWM.

### 19.4.2.1 Input Capture Mode

With the input capture function, the TPM can capture the time at which an external event occurs. When an active edge occurs on the pin of an input capture channel, the TPM latches the contents of the TPM counter into the channel-value registers (TPMxCnVH:TPMxCnVL). Rising edges, falling edges, or any edge is chosen as the active edge that triggers an input capture.

In input capture mode, the TPMxCnVH and TPMxCnVL registers are read only.

When either half of the 16-bit capture register is read, the other half is latched into a buffer to support coherent 16-bit accesses in big-endian or little-endian order. The coherency sequence can be manually reset by writing to TPMxCnSC.

An input capture event sets a flag bit (CHnF) that optionally generates a CPU interrupt request.

While in BDM, the input capture function works as configured. When an external event occurs, the TPM latches the contents of the TPM counter (frozen because of the BDM mode) into the channel value registers and sets the flag bit.

### 19.4.2.2 Output Compare Mode

With the output compare function, the TPM can generate timed pulses with programmable position, polarity, duration, and frequency. When the counter reaches the value in TPMxCnVH:TPMxCnVL registers of an output compare channel, the TPM can set, clear, or toggle the channel pin.

Writes to any of TPMxChnVH and TPMxChnVL registers actually write to buffer registers. In output compare mode, the TPMxChnVH:TPMxChnVL registers are updated with the value of their write buffer only after both bytes were written and according to the value of CLKSB:CLKSA bits:

- If CLKSB and CLKSA are cleared, the registers are updated when the second byte is written
- If CLKSB and CLKSA are not cleared, the registers are updated at the next change of the TPM counter (end of the prescaler counting) after the second byte is written.

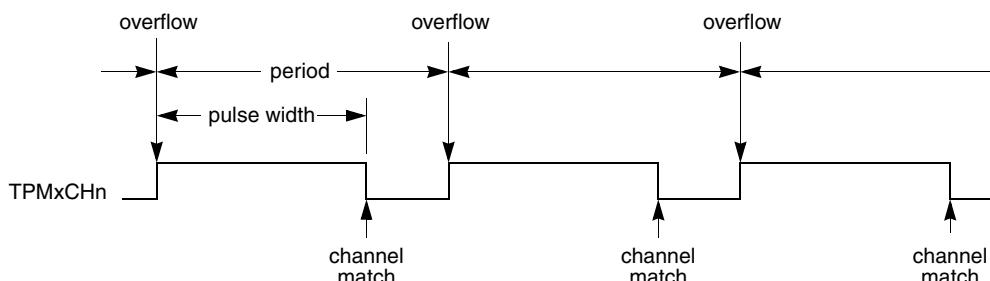
The coherency sequence can be manually reset by writing to the channel status/control register (TPMxChnSC).

An output compare event sets a flag bit (CHnF) that optionally generates a CPU interrupt request.

#### 19.4.2.3 Edge-Aligned PWM Mode

This type of PWM output uses the normal up-counting mode of the timer counter (CPWMS=0) and can be used when other channels in the same TPM are configured for input capture or output compare functions. The period of this PWM signal is determined by the value of the modulus register (TPMxMODH:TPMxMODL) plus 1. The duty cycle is determined by the value of the timer channel register (TPMxChnVH:TPMxChnVL). The polarity of this PWM signal is determined by ELSnA bit. 0% and 100% duty cycle cases are possible.

The time between the modulus overflow and the channel match value (TPMxChnVH:TPMxChnVL) is the pulse width or duty cycle (Figure 19-14). If ELSnA is cleared, the counter overflow forces the PWM signal high, and the channel match forces the PWM signal low. If ELSnA is set, the counter overflow forces the PWM signal low, and the channel match forces the PWM signal high.



**Figure 19-14. EPWM period and pulse width (ELSnA=0)**

When the channel value register is set to 0x0000, the duty cycle is 0%. A 100% duty cycle is achieved by setting the timer-channel register (TPMxChnVH:TPMxChnVL) to a value greater than the modulus setting. This implies that the modulus setting must be less than 0xFFFF in order to get 100% duty cycle.

The timer channel registers are buffered to ensure coherent 16-bit updates and to avoid unexpected PWM pulse widths. Writes to any of the registers TPMxChnVH and TPMxChnVL actually write to buffer registers. In edge-aligned PWM mode, the TPMxChnVH:TPMxChnVL registers are updated with the value of their write buffer according to the value of CLKSB:CLKSA bits:

- If CLKSB and CLKSA are cleared, the registers are updated when the second byte is written
- If CLKSB and CLKSA are not cleared, the registers are updated after both bytes were written, and the TPM counter changes from (TPMxMODH:TPMxMODL – 1) to

(TPMxMODH:TPMxMODL). If the TPM counter is a free-running counter, the update is made when the TPM counter changes from 0xFFFFE to 0xFFFF.

#### 19.4.2.4 Center-Aligned PWM Mode

This type of PWM output uses the up/down counting mode of the timer counter (CPWMS=1). The channel match value in TPMxCnVH:TPMxCnVL determines the pulse width (duty cycle) of the PWM signal while the period is determined by the value in TPMxMODH:TPMxMODL. TPMxMODH:TPMxMODL must be kept in the range of 0x0001 to 0x7FFF because values outside this range can produce ambiguous results. ELSnA determines the polarity of the CPWM signal.

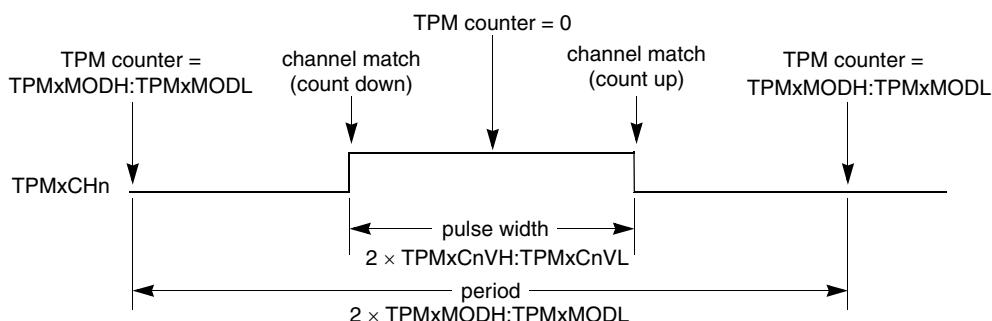
$$\text{pulse width} = 2 \times (\text{TPMxCnVH:TPMxCnVL})$$

$$\text{period} = 2 \times (\text{TPMxMODH:TPMxMODL}); \text{TPMxMODH:TPMxMODL} = 0x0001\text{--}0x7FFF$$

If TPMxCnVH:TPMxCnVL is zero or negative (bit 15 set), the duty cycle is 0%. If TPMxCnVH:TPMxCnVL is a positive value (bit 15 clear) and is greater than the non-zero modulus setting, the duty cycle is 100% because the channel match never occurs. This implies the usable range of periods set by the modulus register is 0x0001 through 0x7FF (0x7FFF if you do not need to generate 100% duty cycle). This is not a significant limitation. The resulting period is much longer than required for normal applications.

All zeros in TPMxMODH:TPMxMODL is a special case that must not be used with center-aligned PWM mode. When CPWMS is cleared, this case corresponds to the counter running free from 0x0000 through 0xFFFF. When CPWMS is set, the counter needs a valid match to the modulus register somewhere other than at 0x0000 in order to change directions from up-counting to down-counting.

The channel match value in the TPM channel registers (times two) determines the pulse width (duty cycle) of the CPWM signal (Figure 19-15). If ELSnA is cleared, a channel match occurring while counting up clears the CPWM output signal and a channel match occurring while counting down sets the output. The counter counts up until it reaches the modulo setting in TPMxMODH:TPMxMODL, then counts down until it reaches zero. This sets the period equal to two times TPMxMODH:TPMxMODL.



**Figure 19-15. CPWM period and pulse width (ELSnA=0)**

Center-aligned PWM outputs typically produce less noise than edge-aligned PWMs because fewer I/O pin transitions are lined up at the same system clock edge. This type of PWM is also required for some types of motor drives.

Input capture, output compare, and edge-aligned PWM functions do not make sense when the counter is operating in up/down counting mode so this implies that all active channels within a TPM must be used in CPWM mode when CPWMS is set.

The timer channel registers are buffered to ensure coherent 16-bit updates and to avoid unexpected PWM pulse widths. Writes to any of the registers TPMxCnVH and TPMxCnVL actually write to buffer registers. In center-aligned PWM mode, the TPMxCnVH:TPMxCnVL registers are updated with the value of their write buffer according to the value of CLKSB:CLKSA bits:

- If CLKSB and CLKSA are cleared, the registers are updated when the second byte is written
- If CLKSB and CLKSA are not cleared, the registers are updated after both bytes were written, and the TPM counter changes from (TPMxMODH:TPMxMODL – 1) to (TPMxMODH:TPMxMODL). If the TPM counter is a free-running counter, the update is made when the TPM counter changes from 0xFFFF to 0xFFFF.

When TPMxCNTH:TPMxCNTL equals TPMxMODH:TPMxMODL, the TPM can optionally generate a TOF interrupt (at the end of this count).

## 19.5 Reset Overview

### 19.5.1 General

The TPM is reset whenever any MCU reset occurs.

### 19.5.2 Description of Reset Operation

Reset clears TPMxSC that disables TPM counter clock and overflow interrupt (TOIE=0). CPWMS, MSnB, MSnA, ELSnB, and ELSnA are all cleared. This configures all TPM channels for input capture operation and the associated pins are not controlled by TPM.

## 19.6 Interrupts

### 19.6.1 General

The TPM generates an optional interrupt for the main counter overflow and an interrupt for each channel. The meaning of channel interrupts depends on each channel's mode of operation. If the channel is configured for input capture, the interrupt flag is set each time the selected input capture edge is recognized. If the channel is configured for output compare or PWM modes, the interrupt flag is set each time the main timer counter matches the value in the 16-bit channel value register.

All TPM interrupts are listed in [Table 19-7](#).

**Table 19-7. Interrupt Summary**

<b>Interrupt</b>	<b>Local Enable</b>	<b>Source</b>	<b>Description</b>
TOF	TOIE	Counter overflow	Set each time the TPM counter reaches its terminal count (at transition to its next count value)
CHnF	CHnIE	Channel event	An input capture event or channel match took place on channel n

The TPM module provides high-true interrupt signals.

## 19.6.2 Description of Interrupt Operation

For each interrupt source in the TPM, a flag bit is set upon recognition of the interrupt condition such as timer overflow, channel input capture, or output compare events. This flag is read (polled) by software to determine that the action has occurred, or an associated enable bit (TOIE or CHnIE) can be set to enable the interrupt generation. While the interrupt enable bit is set, the interrupt is generated whenever the associated interrupt flag is set. Software must perform a sequence of steps to clear the interrupt flag before returning from the interrupt-service routine.

TPM interrupt flags are cleared by a two-step process including a read of the flag bit while it is set followed by a write of zero to the bit. If a new event is detected between these two steps, the sequence is reset and the interrupt flag remains set after the second step to avoid the possibility of missing the new event.

### 19.6.2.1 Timer Overflow Interrupt (TOF) Description

The meaning and details of operation for TOF interrupts varies slightly depending upon the mode of operation of the TPM system (general purpose timing functions versus center-aligned PWM operation). The flag is cleared by the two step sequence described above.

#### 19.6.2.1.1 Normal Case

When CPWMS is cleared, TOF is set when the timer counter changes from the terminal count (the value in the modulo register) to 0x0000. If the TPM counter is a free-running counter, the update is made when the TPM counter changes from 0xFFFF to 0x0000.

#### 19.6.2.1.2 Center-Aligned PWM Case

When CPWMS is set, TOF is set when the timer counter changes direction from up-counting to down-counting at the end of the terminal count (the value in the modulo register).

### 19.6.2.2 Channel Event Interrupt Description

The meaning of channel interrupts depends on the channel's current mode (input capture, output compare, edge-aligned PWM, or center-aligned PWM).

### 19.6.2.2.1 Input Capture Events

When a channel is configured as an input capture channel, the ELSnB:ELSnA bits select if channel pin is not controlled by TPM, rising edges, falling edges, or any edge as the edge that triggers an input capture event. When the selected edge is detected, the interrupt flag is set. The flag is cleared by the two-step sequence described in [Section 19.6.2, “Description of Interrupt Operation.”](#)

### 19.6.2.2.2 Output Compare Events

When a channel is configured as an output compare channel, the interrupt flag is set each time the main timer counter matches the 16-bit value in the channel value register. The flag is cleared by the two-step sequence described in [Section 19.6.2, “Description of Interrupt Operation.”](#)

### 19.6.2.2.3 PWM End-of-Duty-Cycle Events

When the channel is configured for edge-aligned PWM, the channel flag is set when the timer counter matches the channel value register that marks the end of the active duty cycle period. When the channel is configured for center-aligned PWM, the timer count matches the channel value register twice during each PWM cycle. In this CPWM case, the channel flag is set at the start and at the end of the active duty cycle period when the timer counter matches the channel value register. The flag is cleared by the two-step sequence described in [Section 19.6.2, “Description of Interrupt Operation.”](#)

# **Chapter 20**

## **Version 1 ColdFire Debug (CF1\_DEBUG)**

### **20.1 Introduction**

This chapter describes the capabilities defined by the Version 1 ColdFire debug architecture. The Version 1 ColdFire core supports BDM functionality using the HCS08's single-pin interface. The traditional 3-pin full-duplex ColdFire BDM serial communication protocol based on 17-bit data packets is replaced with the HCS08 debug protocol where all communication is based on an 8-bit data packet using a single package pin (BKGD).

An on-chip trace buffer allows a stream of compressed processor execution status packets to be recorded for subsequent retrieval to provide program (and partial data) trace capabilities.

The following sections in this chapter provide details on the BKGD pin, the background debug serial interface controller (BDC), a standard 6-pin BDM connector, the BDM command set as well as real-time debug and trace capabilities. The V1 definition supports revision B+ (DEBUG\_B+) of the ColdFire debug architecture.

A simplified block diagram of the V1 core including the processor and debug module is shown in [Figure 20-1](#).

#### **NOTE**

Use pin mux control registers from [Section 2.3, “Pin Mux Controls”](#) to assign the debug signals to the device's package pins.

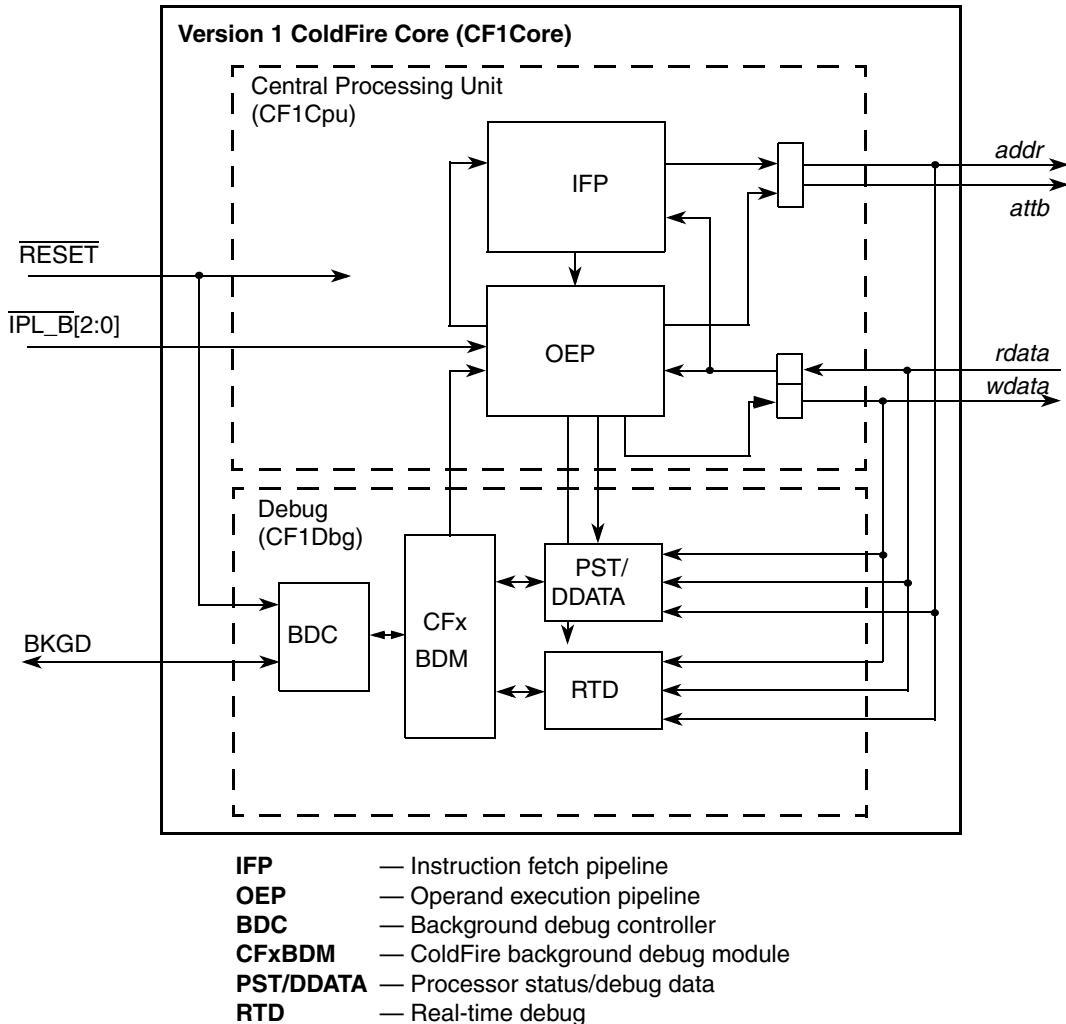


Figure 20-1. Simplified Version 1 ColdFire Core Block Diagram

### 20.1.1 Overview

Debug support is divided into three areas:

- Background debug mode (BDM)—Provides low-level debugging in the ColdFire processor core. In BDM, the processor core is halted and a variety of commands can be sent to the processor to access memory, registers, and peripherals. The external emulator uses a one-pin serial communication protocol. See [Section 20.4.1, “Background Debug Mode \(BDM\)”](#).
- Real-time debug support—Use of the full BDM command set requires the processor to be halted, which many real-time embedded applications cannot support. The core includes a variety of internal breakpoint registers which can be configured to trigger and generate a special interrupt. The resulting debug interrupt lets 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. The external development system can then access the saved data, because the hardware supports

concurrent operation of the processor and BDM-initiated memory commands. In addition, the option is provided to allow interrupts to occur. See [Section 20.4.2, “Real-Time Debug Support”](#).

- Program trace support—The ability to determine the dynamic execution path through an application is fundamental for debugging. The V1 solution implements a trace buffer that records processor execution status and data, which can be subsequently accessed by the external emulator system to provide program (and optional partial data) trace information. See [Section 20.4.3, “Trace Support”](#).

There are two fields in debug registers which provide revision information: the hardware revision level in CSR and the 1-pin debug hardware revision level in CSR2. [Table 20-1](#) summarizes the various debug revisions.

**Table 20-1. Debug Revision Summary**

Revision	CSR[HRL]	CSR2[D1HRL]	Enhancements
A	0000	N/A	Initial ColdFire debug definition
B	0001	N/A	BDM command execution does not affect hardware breakpoint logic Added BDM address attribute register (BAAR) <del>BKPT</del> configurable interrupt (CSR[BKD]) Level 1 and level 2 triggers on OR condition, in addition to AND SYNC_PC command to display the processor's current PC
B+	1001	N/A	Added 3 PC breakpoint registers PBR1–3
CF1_B+	1001	0001	Converted to HCS08 1-pin BDM serial interface Added PST compression and on-chip PST/DDATA buffer for program trace

## 20.1.2 Features

The Version 1 ColdFire debug definition supports the following features:

- Classic ColdFire DEBUG\_B+ functionality mapped into the single-pin BDM interface
- Real time debug support, with 6 hardware breakpoints (4 PC, 1 address pair and 1 data) that can be configured into a 1- or 2-level trigger with a programmable response (processor halt or interrupt)
- Capture of compressed processor status and debug data into on-chip trace buffer provides program (and optional slave bus data) trace capabilities
- On-chip trace buffer provides programmable start/stop recording conditions plus support for obtrusive or PC-profiling modes
- Debug resources are accessible via single-pin BDM interface or the privileged WDEBUG instruction from the core

## 20.1.3 Modes of Operations

V1 ColdFire devices typically implement a number of modes of operation, including run, wait, and stop modes. Additionally, the operation of the core’s debug module is highly dependent on a number of chip configurations which determine its operating state.

When operating in secure mode, as defined by a 2-bit field in the flash memory examined at reset, BDM access to debug resources is extremely restricted. It is possible to tell that the device has been secured, and

to clear security, which involves mass erasing the on-chip flash memory. No other debug access is allowed. Secure mode can be used in conjunction with each of the wait and stop low-power modes.

If the BDM interface is not enabled, access to the debug resources is limited in the same manner as a secure device.

If the device is not secure and the BDM interface is enabled (XCSR[ENBDM] is set), the device is operating in debug mode and additional resources are available via the BDM interface. In this mode, the status of the processor (running, stopped, or halted) determines which BDM commands may be used.

Debug mode functions are managed through the background debug controller (BDC) in the Version 1 ColdFire core. The BDC provides the means for analyzing MCU operation during software development.

BDM commands can be classified into three types as shown in [Table 20-2](#).

**Table 20-2. BDM Command Types**

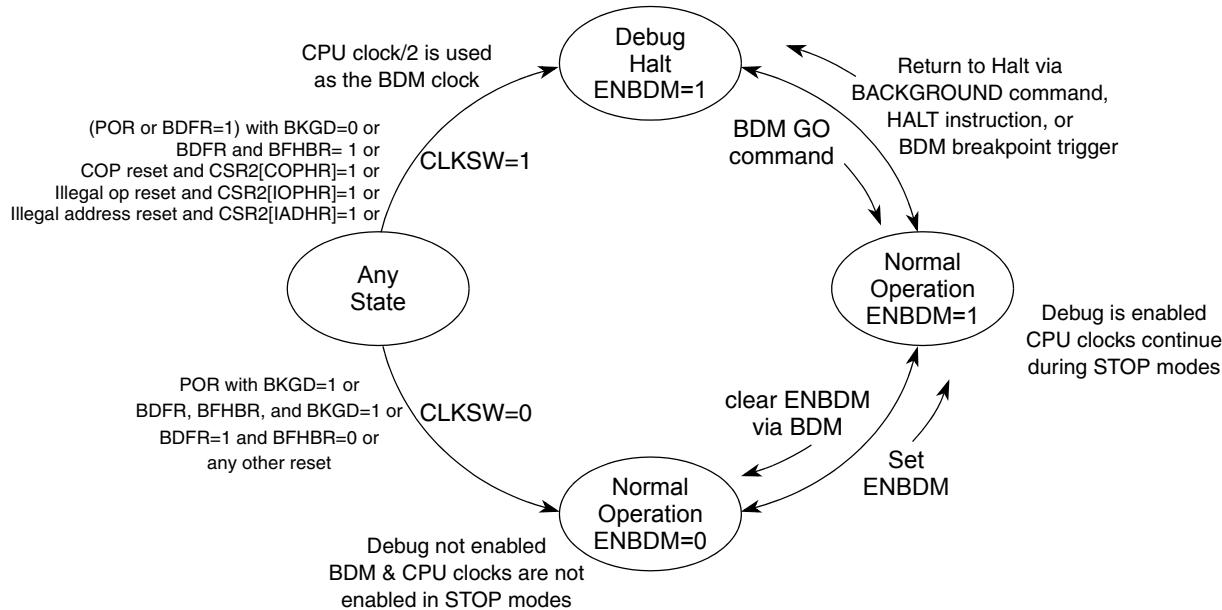
Command Type	Flash Secure?	BDM?	Core Status	Command Set
Always-available	Secure or Unsecure	Enabled or Disabled	—	<ul style="list-style-type: none"> <li>Read/write access to XCSR[31–24], CSR2[31–24], CSR3[31–24]</li> </ul>
Non-intrusive	Unsecure	Enabled	Run, Halt	<ul style="list-style-type: none"> <li>Memory access</li> <li>Memory access with status</li> <li>Debug register access</li> <li>BACKGROUND</li> </ul>
Active background	Unsecure	Enabled	Halt	<ul style="list-style-type: none"> <li>Read or write CPU registers (also available in stop mode)</li> <li>Single-step the application</li> <li>Exit halt mode to return to the application program (GO)</li> </ul>

For more information on these three BDM command classifications, see [Section 20.4.1.5, “BDM Command Set Summary.”](#)

The core’s halt mode is entered in a number of ways:

- The BKGD pin is low during POR
- The BKGD pin is low immediately after a BDM-initiated force reset (see CSR2[BDFR] in [Section 20.3.3, “Configuration/Status Register 2 \(CSR2\),”](#) for details)
- A background debug force reset occurs (CSR2[BDFR] is set) and CSR2[BFHBR] is set
- A computer operating properly reset occurs and CSR2[COPHR] is set
- An illegal operand reset occurs and CSR2[IOPHR] is set
- An illegal address reset occurs and CSR2[IADHR] is set
- A BACKGROUND command is received through the BKGD pin. If necessary, this wakes the device from STOP/WAIT modes.
- A properly-enabled (XCSR[ENBDM] is set) HALT instruction is executed
- Encountering a BDM breakpoint and the trigger response is programmed to generate a halt
- Reaching a PSTB trace buffer full condition when operating in an obtrusive recording mode (CSR2[PSTBRM] is set to 01 or 11)

While in halt mode, the core waits for serial background commands rather than executing instructions from the application program.



**Figure 20-2. Debug Modes State Transition Diagram**

Figure 20-2 contains a simplified view of the V1 ColdFire debug mode states. The XCSR[CLKSW] bit controls the BDC clock source. When CLKSW is set, the BDC serial clock frequency is half the CPU clock. When CLKSW is cleared, the BDC serial clock is supplied from an alternate clock source.

The ENBDM bit determines if the device can be placed in halt mode, if the core and BDC serial clocks continue to run in STOP modes, and if the regulator can be placed into standby mode. Again, if booting to halt mode, XCSR[ENBDM, CLKSW] are automatically set.

If ENBDM is cleared, the ColdFire core treats the HALT instruction as an illegal instruction and generates a reset (if CPUCR[IRD] is cleared) or an exception (if CPUCR[IRD] is set) if execution is attempted.

If XCSR[ENBDM] is set, the device can be restarted from STOP/WAIT via the BDM interface.

## 20.2 External Signal Descriptions

Table 20-3 describes the debug module's 1-pin external signal (BKGD). A standard 6-pin debug connector is shown in Section 20.4.4, “Freescale-Recommended BDM Pinout”.

**Table 20-3. Debug Module Signals**

Signal	Description
Background Debug (BKGD)	Single-wire background debug interface pin. The primary function of this pin is for bidirectional serial communication of background debug mode commands and data. During reset, this pin selects between starting in active background (halt) mode or starting the application program. This pin also requests a timed sync response pulse to allow a host development tool to determine the correct clock frequency for background debug serial communications.

## 20.3 Memory Map/Register Definition

In addition to the BDM commands that provide access to the processor's registers and the memory subsystem, the debug module contains a number of registers. Most of these registers (all except the PST/DDATA trace buffer) are also accessible (write-only) from the processor's supervisor programming model by executing the WDEBUG instruction. Thus, the breakpoint hardware in the debug module can be read (certain registers) or written by the external development system using the serial debug interface or written by the operating system running on the processor core. Software is responsible for guaranteeing that accesses to these resources are serialized and logically consistent. The 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 during the processor's execution of the WDEBUG instruction to configure debug module registers or the resulting behavior is undefined.

These registers, shown in [Table 20-4](#), are treated as 32-bit quantities regardless of the number of implemented bits and unimplemented bits are reserved and must be cleared. These registers are also accessed through the BDM port by the commands, WRITE\_DREG and READ\_DREG, described in [Section 20.4.1.5, “BDM Command Set Summary.”](#) These commands contain a 5-bit field, DRc, that specifies the register, as shown in [Table 20-4](#).

**Table 20-4. Debug Module Memory Map**

DRc	Register Name	Width (bits)	Access	Reset Value	Section/ Page
0x00	Configuration/status register (CSR)	32	R/W (BDM), W (CPU)	0x0090_0000	<a href="#">20.3.1/20-7</a>
0x01	Extended Configuration/Status Register (XCSR)	32	R/W <sup>1</sup> (BDM), W (CPU)	0x0000_0000	<a href="#">20.3.2/20-10</a>
0x02	Configuration/Status Register 2 (CSR2)	32	R/W <sup>1</sup> (BDM), W (CPU)	See Section	<a href="#">20.3.3/20-13</a>
0x03	Configuration/Status Register 3 (CSR3)	32 <sup>2</sup>	R/W <sup>1</sup> (BDM), W (CPU)	0x0000_0000	<a href="#">20.3.4/20-16</a>
0x05	BDM address attribute register (BAAR)	32 <sup>2</sup>	W	0x0000_0005	<a href="#">20.3.5/20-17</a>
0x06	Address attribute trigger register (AATR)	32 <sup>2</sup>	W	0x0000_0005	<a href="#">20.3.6/20-18</a>
0x07	Trigger definition register (TDR)	32	W	0x0000_0000	<a href="#">20.3.7/20-19</a>
0x08	PC breakpoint register 0 (PBR0)	32	W	Undefined, Unaffected	<a href="#">20.3.8/20-22</a>
0x09	PC breakpoint mask register (PBMR)	32	W	Undefined, Unaffected	<a href="#">20.3.8/20-22</a>
0x0C	Address breakpoint high register (ABHR)	32	W	Undefined, Unaffected	<a href="#">20.3.9/20-24</a>
0x0D	Address breakpoint low register (ABLR)	32	W	0x0000_0000	<a href="#">20.3.9/20-24</a>
0x0E	Data breakpoint register (DBR)	32	W	0x0000_0000	<a href="#">20.3.10/20-25</a>
0x0F	Data breakpoint mask register (DBMR)	32	W	0x0000_0000	<a href="#">20.3.10/20-25</a>

**Table 20-4. Debug Module Memory Map (continued)**

DRc	Register Name	Width (bits)	Access	Reset Value	Section/Page
0x18	PC breakpoint register 1 (PBR1)	32	W	PBR1[0] = 0	<a href="#">20.3.8/20-22</a>
0x1A	PC breakpoint register 2 (PBR2)	32	W	PBR2[0] = 0	<a href="#">20.3.8/20-22</a>
0x1B	PC breakpoint register 3 (PBR3)	32	W	PBR3[0] = 0	<a href="#">20.3.8/20-22</a>
—	PST Trace Buffer $n$ (PSTB $n$ ); $n = 0\text{--}11$ (0xB)	32	R (BDM) <sup>3</sup>	Undefined, Unaffected	<a href="#">20.4.1.5.12/20-46</a>

<sup>1</sup> The most significant bytes of the XCSR, CSR2, and CSR3 registers support special control functions and are writeable via BDM using the WRITE\_XCSR\_BYTE, WRITE\_CSR2\_BYTE, and WRITE\_CSR3\_BYTE commands. They can be read from BDM using the READ\_XCSR\_BYTE, READ\_CSR2\_BYTE, and READ\_CSR3\_BYTE commands. These 3 registers, along with the CSR, can also be referenced as 32-bit quantities using the BDM READ\_DREG and WRITE\_DREG commands, but the WRITE\_DREG command only writes bits 23–0 of these three registers.

<sup>2</sup> Each debug register is accessed as a 32-bit value; undefined fields are reserved and must be cleared.

<sup>3</sup> The contents of the PST trace buffer is only read from BDM (32 bits per access) using READ\_PSTB commands.

### NOTE

Debug control registers can be written by the external development system or the CPU through the WDEBUG instruction. These control registers are write-only from the programming model and they can be written through the BDM port using the WRITE\_DREG command. In addition, the four configuration/status registers (CSR, XCSR, CSR2, CSR3) can be read through the BDM port using the READ\_DREG command.

The ColdFire debug architecture supports a number of hardware breakpoint registers that can be configured into single- or double-level triggers based on the PC or operand address ranges with an optional inclusion of specific data values. The triggers can be configured to halt the processor or generate a debug interrupt exception. Additionally, these same breakpoint registers can be used to specify start/stop conditions for recording in the PST trace buffer.

The core includes four PC breakpoint triggers and a set of operand address breakpoint triggers with two independent address registers (to allow specification of a range) and an optional data breakpoint with masking capabilities. Core breakpoint triggers are accessible through the serial BDM interface or written through the supervisor programming model using the WDEBUG instruction.

### 20.3.1 Configuration/Status Register (CSR)

CSR defines the debug configuration for the processor and memory subsystem and contains status information from the breakpoint logic. CSR is accessible from the programming model using the WDEBUG instruction and through the BDM port using the READ\_DREG and WRITE\_DREG commands.

DRc[4:0]: 0x00 (CSR)

Access: Supervisor write-only  
BDM read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	BSTAT				FOF	TRG	HALT	BKPT	HRL				0	BKD	0	IPW
W																
Reset	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0
R	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
W	0	TRC	0	DDC	UHE	BTB		0	NPL	IPI	SSM		0	0	FID	DDH
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 20-3. Configuration/Status Register (CSR)

Table 20-5. CSR Field Descriptions

Field	Description
31–28 BSTAT	Breakpoint status. Provides read-only status (from the BDM port only) information concerning hardware breakpoints. BSTAT is cleared by a TDR write, by a CSR read when a level-2 breakpoint is triggered, or a level-1 breakpoint is triggered and the level-2 breakpoint is disabled. The PSTB value that follows the PSTB entry of 0x1B is $0x20 + (2 \times \text{BSTAT})$ . 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. Indicates a catastrophic halt occurred and forced entry into BDM. FOF is cleared by reset or when CSR is read (from the BDM port only).
26 TRG	Hardware breakpoint trigger. Indicates a hardware breakpoint halted the processor core and forced entry into BDM. Reset, the debug GO command, or reading CSR (from the BDM port only) clears TRG.
25 HALT	Processor halt. Indicates the processor executed a HALT and forced entry into BDM. Reset, the debug GO command, or reading CSR (from the BDM port only) clears HALT.
24 BKPT	Breakpoint assert. Indicates when either: <ul style="list-style-type: none"> <li>• The BKPT input was asserted,</li> <li>• BDM BACKGROUND command received, or</li> <li>• The PSTB halt on full condition, CSR2[PSTBH], sets.</li> </ul> This forces the processor into a BDM halt. Reset, the debug GO command, or reading CSR (from the BDM port only) clears BKPT.
23–20 HRL	Hardware revision level. Indicates, from the BDM port only, the level of debug module functionality. An emulator can use this information to identify the level of functionality supported. 0000 Revision A 0001 Revision B 0010 Revision C 0011 Revision D 1001 Revision B+ (The value used for this device) 1011 Revision D+
19	Reserved, must be cleared.

**Table 20-5. CSR Field Descriptions (continued)**

Field	Description
18 BKD	Breakpoint disable. Disables the BACKGROUND command functionality, and allows the execution of the BACKGROUND command to generate a debug interrupt. 0 Normal operation 1 The receipt of a BDM BACKGROUND command signals a debug interrupt to the ColdFire core. The processor makes this interrupt request pending until the next sample point occurs, when the exception is initiated. In the ColdFire architecture, the interrupt sample point occurs once per instruction. There is no support for nesting debug interrupts.
17	Reserved, must be cleared.
16 IPW	Inhibit processor writes. Inhibits processor-initiated writes to the debug module's programming model registers. IPW can be modified only by commands from the BDM interface.
15	Reserved, must be cleared.
14 TRC	Force emulation mode on trace exception. 0 Processor enters supervisor mode. 1 Processor enters emulator mode when a trace exception occurs.
13	Reserved, must be cleared.
12–11 DDC	Debug data control. Controls peripheral bus operand data capture for DDATA, which displays the number of bytes defined by the operand reference size (a marker) 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 clock cycles). See <a href="#">Table 20-26</a> . A non-zero value enables partial data trace capabilities. 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. The core must be operating with XCSR[ENBDM] set to execute any HALT instruction, else the instruction is treated as an illegal opcode. 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. See <a href="#">Section 20.4.3.1, "Begin Execution of Taken Branch (PST = 0x05)"</a> . 00 No target address capture 01 Lower 2 bytes of the target address 1x Lower 3 bytes of the target address
7	Reserved, must be cleared.
6 NPL	Non-pipelined mode. Determines if the core operates in pipelined mode. 0 Pipelined mode 1 Non-pipelined mode. The processor effectively executes one instruction at a time with no overlap. This typically adds five cycles to the execution time of each instruction. Given an average execution latency of ~2 cycles per instruction, throughput in non-pipeline mode would be ~7 cycles per instruction, approximately 25% - 33% 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, these triggers are always reported before the next instruction begins execution and trigger reporting can be considered precise.

**Table 20-5. CSR Field Descriptions (continued)**

Field	Description
5 IPI	Ignore pending interrupts when in single-step mode. 0 Core services any pending interrupt requests signalled while in single-step mode. 1 Core ignores any pending interrupt requests signalled while in single-step mode.
4 SSM	Single-step mode enable. 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.
3–2	Reserved, must be cleared.
1 FID	Force <i>ipg_debug</i> . The core generates this output to the device, signaling it is in debug mode. 0 Do not force the assertion of <i>ipg_debug</i> 1 Force the assertion of <i>ipg_debug</i>
0 DDH	Disable <i>ipg_debug</i> due to a halt condition. The core generates an output to the other modules in the device, signaling it is in debug mode. By default, this output signal is asserted when the core halts. 0 Assert <i>ipg_debug</i> if the core is halted 1 Negate <i>ipg_debug</i> due to the core being halted

### 20.3.2 Extended Configuration/Status Register (XCSR)

The 32-bit XCSR is partitioned into two sections: the upper byte contains status and command bits always accessible to the BDM interface, even if debug mode is disabled. This status byte is also known as XCSR\_SB. The lower 24 bits contain fields related to the generation of automatic SYNC\_PC commands, which can be used to periodically capture and display the current program counter (PC) in the PST trace buffer (if properly configured).

There are multiple ways to reference the XCSR. They are summarized in [Table 20-6](#).

**Table 20-6. XCSR Reference Summary**

Method	Reference Details
READ_XCSR_BYTE	Reads XCSR[31–24] from the BDM interface. Available in all modes.
WRITE_XCSR_BYTE	Writes XCSR[31–24] from the BDM interface. Available in all modes.
READ_DREG	Reads XCSR[31–0] from the BDM interface. Classified as a non-intrusive BDM command.
WRITE_DREG	Writes XCSR[23–0] from the BDM interface. Classified as a non-intrusive BDM command.
WDEBUG instruction	Writes XCSR[23–0] during the core's execution of WDEBUG instruction. This instruction is a privileged supervisor-mode instruction.

DRc: 0x01 (XCSR)

Access: Supervisor write-only  
BDM read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	CPU HALT	CPU STOP	CSTAT			CLK SW	SEC	EN BDM	0	0	0	0	0	0	0	0
W			ESEQC			ERASE										
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	APCSC	APC ENB
W															0	0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 20-4. Extended Configuration/Status Register (XCSR)

Table 20-7. XCSR Field Descriptions

Field	Description														
31 CPUHALT	Indicates that the CPU is in the halt state. The CPU state may be running, stopped, or halted, which is determined by the CPUHALT and CPUSTOP bits as shown below.														
	<table border="1"> <thead> <tr> <th>XCSR [CPUHALT]</th> <th>XCSR [CPUSTOP]</th> <th>CPU State</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>Running</td> </tr> <tr> <td>0</td> <td>1</td> <td>Stopped</td> </tr> <tr> <td>1</td> <td>0</td> <td>Halted</td> </tr> </tbody> </table>			XCSR [CPUHALT]	XCSR [CPUSTOP]	CPU State	0	0	Running	0	1	Stopped	1	0	Halted
XCSR [CPUHALT]	XCSR [CPUSTOP]	CPU State													
0	0	Running													
0	1	Stopped													
1	0	Halted													
30 CPUSTOP	Indicates that the CPU is in the stop state. The CPU state may be running, stopped, or halted, which is determined by the CPUHALT and CPUSTOP bits as shown in the CPUHALT bit description.														

**Table 20-7. XCSR Field Descriptions (continued)**

Field	Description
29–27 CSTAT (R) ESEQC (W)	<p>During reads, indicates the BDM command status.</p> <p>000 Command done, no errors 001 Command done, data invalid 01x Command done, illegal 1xx Command busy, overrun</p> <p>If an overrun is detected (CSTAT = 1xx), the following sequence is suggested to clear the source of the error:</p> <ol style="list-style-type: none"> <li>1. Issue a SYNC command to reset the BDC channel.</li> <li>2. The host issues a BDM NOP command.</li> <li>3. The host checks the channel status using a READ_XCSR_BYTE command.</li> <li>4. If XCSR[CSTAT] = 000           <ul style="list-style-type: none"> <li>then status is okay; proceed</li> <li>else               <ul style="list-style-type: none"> <li>Halt the CPU with a BDM BACKGROUND command</li> <li>Repeat steps 1,2,3</li> <li>If XCSR[CSTAT] ≠ 000, then reset device</li> </ul> </li> </ul> </li> </ol> <p>During writes, the ESEQC field is used for the erase sequence control during flash programming. ERASE must also be set for this bit to have an effect.</p> <p>000 User mass erase Else Reserved</p> <p><b>Note:</b> See the Memory chapter for a detailed description of the algorithm for clearing security.</p>
26 CLKSW	<p>Select source for serial BDC communication clock.</p> <p>0 Alternate, asynchronous BDC clock, typically 10 MHz 1 Synchronous bus clock (CPU clock divided by 2)</p> <p>The initial state of the XCSR[CLKSW] bit is loaded by the hardware in response to certain reset events and the state of the BKGD pin as described in <a href="#">Figure 20-2</a>.</p>
25 SEC (R) ERASE (W)	<p>The read value of this bit typically defines the status of the flash security field.</p> <p>0 Flash security is disabled 1 Flash security is enabled</p> <p>In addition, the SEC bit is context-sensitive during reads. After a mass-erase sequence has been initiated by BDM, it acts as a flash busy flag. When the erase operation is complete and the bit is cleared, it returns to reflect the status of the chip security.</p> <p>0 Flash is not busy performing a BDM mass-erase sequence 1 Flash is busy performing a BDM mass-erase sequence</p> <p>During writes, this bit qualifies XCSR[ESEQC] for the write modes shown in the ESEQC field description.</p> <p>0 Do not perform a mass-erase of the flash. 1 Perform a mass-erase of the flash, using the sequence specified in the XCSR[ESEQC] field.</p>
24 ENBDM	Enable BDM.
23–3	Reserved for future use by the debug module, must be cleared.

**Table 20-7. XCSR Field Descriptions (continued)**

Field	Description																																							
2–1 APCSC	<p>Automatic PC synchronization control. Determines the periodic interval of PC address captures, if XCSR[APCENB] is set. When the selected interval is reached, a SYNC_PC command is sent to the ColdFire CPU. For more information on the SYNC_PC operation, see the APCENB description.</p> <p>The chosen frequency depends on CSR2[APCDIV16] as shown in the equation and table below:</p> $\text{PC address capture period} = \frac{2^{(\text{APCSC} + 1)} \times 1024}{16^{\text{APCDIV16}}} \quad \text{Eqn. 20-1}$ <table border="1"> <thead> <tr> <th>XCSR [APCENB]</th> <th>CSR2 [APCDIV16]</th> <th>XCSR [APCSC]</th> <th>SYNC_PC Interval</th> </tr> </thead> <tbody> <tr><td>1</td><td>0</td><td>00</td><td>2048 cycles</td></tr> <tr><td>1</td><td>0</td><td>01</td><td>4096 cycles</td></tr> <tr><td>1</td><td>0</td><td>10</td><td>8192 cycles</td></tr> <tr><td>1</td><td>0</td><td>11</td><td>16384 cycles</td></tr> <tr><td>1</td><td>1</td><td>00</td><td>128 cycles</td></tr> <tr><td>1</td><td>1</td><td>01</td><td>256 cycles</td></tr> <tr><td>1</td><td>1</td><td>10</td><td>512 cycles</td></tr> <tr><td>1</td><td>1</td><td>11</td><td>1024 cycles</td></tr> </tbody> </table>				XCSR [APCENB]	CSR2 [APCDIV16]	XCSR [APCSC]	SYNC_PC Interval	1	0	00	2048 cycles	1	0	01	4096 cycles	1	0	10	8192 cycles	1	0	11	16384 cycles	1	1	00	128 cycles	1	1	01	256 cycles	1	1	10	512 cycles	1	1	11	1024 cycles
XCSR [APCENB]	CSR2 [APCDIV16]	XCSR [APCSC]	SYNC_PC Interval																																					
1	0	00	2048 cycles																																					
1	0	01	4096 cycles																																					
1	0	10	8192 cycles																																					
1	0	11	16384 cycles																																					
1	1	00	128 cycles																																					
1	1	01	256 cycles																																					
1	1	10	512 cycles																																					
1	1	11	1024 cycles																																					
0 APCENB	<p>Automatic PC synchronization enable. Enables the periodic output of the PC which can be used for PST/DDATA trace synchronization and code profiling.</p> <p>As described in XCSR[APCSC], when the enabled periodic timer expires, a SYNC_PC command is sent to the ColdFire CPU which generates a forced instruction fetch of the next instruction. The PST/DDATA module captures the target address as defined by CSR[9] (two bytes if CSR[9] is cleared, three bytes if CSR[9] is set). This produces a PST sequence of the PST marker indicating a 2- or 3-byte address, followed by the captured instruction address.</p> <p>0 Automatic PC synchronization disabled 1 Automatic PC synchronization enabled</p>																																							

### 20.3.3 Configuration/Status Register 2 (CSR2)

The 32-bit CSR2 is partitioned into two sections. The upper byte contains status and configuration bits always accessible to the BDM interface, even if debug mode is disabled. The lower 24 bits contain fields related to the configuration of the PST trace buffer (PSTB).

There are multiple ways to reference CSR2. They are summarized in [Table 20-8](#).

**Table 20-8. CSR2 Reference Summary**

Method	Reference Details
READ_CSR2_BYTE	Reads CSR2[31–24] from the BDM interface. Available in all modes.
WRITE_CSR2_BYTE	Writes CSR2[31–24] from the BDM interface. Available in all modes.
READ_DREG	Reads CSR2[31–0] from the BDM interface. Classified as a non-intrusive BDM command.

Table 20-8. CSR2 Reference Summary (continued)

Method	Reference Details
WRITE_DREG	Writes CSR2[23–0] from the BDM interface. Classified as a non-intrusive BDM command.
WDEBUG Instruction	Writes CSR2[23–0] during the core's execution of WDEBUG instruction. This instruction is a privileged supervisor-mode instruction.

DRc: 0x02 (CSR2)

Access: Supervisor read-only  
BDM read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	PSTBP	0	COP	IOP	IAD	0	BFHBR	0	PSTBH	PSTBST	0		D1HRL			
W			HR	HR	HR			BDFR								
Power-on Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Other Reset	0	0	u	u	u	0	u	0	0	0	0	0	0	0	0	1
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	PSTBWA								0	APC	0		PSTBRM		PSTBSS	
W									PSTBR	DIV16						
Reset	Unaffected and Undefined								0	0	0	0	0	0	0	0

Figure 20-5. Configuration/Status Register 2 (CSR2)

Table 20-9. CSR2 Field Descriptions

Field	Description
31 PSTBP	PST buffer stop. Signals if a PST buffer stop condition has been reached. 0 A PST trace buffer stop condition has not been reached 1 A PST trace buffer stop condition has been reached
30	Reserved, must be cleared.
29 COPHR	Computer operating properly halt after reset. Determines operation of the device after a COP reset. This bit is cleared after a power-on reset and is unaffected by any other reset. 0 After a computer operating properly reset, the device immediately enters normal operation mode. 1 A computer operating properly reset immediately halts the device (as if the BKGD pin was held low after a power-on reset). <b>Note:</b> This bit may only be changed if XCSR[ENBDM] is set and the flash is unsecure.
28 IOPHR	Illegal operation halt after reset. Determines operation of the device after an illegal operation reset. This bit is cleared after a power-on reset and is unaffected by any other reset. 0 After the device has an illegal operation reset, the device immediately enters normal operation mode. 1 An illegal operation reset immediately halts the device (as if the BKGD pin was held low after a power-on reset). <b>Note:</b> This bit may only be changed if XCSR[ENBDM] is set and the flash is unsecure.
27 IADHR	Illegal address halt after reset. Determines operation of the device after an illegal address reset. This bit is cleared after a power-on reset and is unaffected by any other reset. 0 After the device has an illegal address reset, the device immediately enters normal operation mode. 1 An illegal address reset immediately halts the device (as if the BKGD pin was held low after a power-on reset). <b>Note:</b> This bit may only be changed if XCSR[ENBDM] is set and the flash is unsecure.

**Table 20-9. CSR2 Field Descriptions (continued)**

Field	Description						
26	Reserved, must be cleared.						
25 BFHBR	BDM force halt on BDM reset. Determines operation of the device after a BDM reset. This bit is cleared after a power-on reset and is unaffected by any other reset. 0 The device enters normal operation mode following a BDM reset. 1 The device enters in halt mode following a BDM reset, as if the BKGD pin was held low after a power-on-reset or standard BDM-initiated reset. <b>Note:</b> This bit can only change state if XCSR[ENBDM] = 1 and the flash is unsecure.						
24 BDFR	Background debug force reset. Forces a BDM reset to the device. This bit always reads as 0 after the reset has been initiated. 0 No reset initiated. 1 Force a BDM reset.						
23 PSTBH	PST trace buffer halt. Indicates if the processor is halted due to the PST trace buffer being full when recording in a obtrusive mode. 0 PST trace buffer not full 1 CPU halted due to PST trace buffer being full in obtrusive mode						
22–21 PSTBST	PST trace buffer state. Indicates the current state of the PST trace buffer recording. 00 PSTB disabled 01 PSTB enabled and waiting for the start condition 10 PSTB enabled, recording and waiting for the stop condition 11 PSTB enabled, completed recording after the stop condition was reached						
20	Reserved, must be cleared.						
19–16 D1HRL	Debug 1-pin hardware revision level. Indicates the hardware revision level of the 1-pin debug module implemented in the ColdFire core. For this device, this field is 0x1.						
15–8 PSTBWA	PST trace buffer write address. Indicates the current write address of the PST trace buffer. The most significant bit of this field is sticky; if set, it remains set until a PST/DDATA reset event occurs. As the ColdFire core inserts PST and DDATA packets into the trace buffer, this field is incremented. The value of the write address defines the next location in the PST trace buffer to be loaded. In other words, the contents of PSTB[PSTBWA-1] is the last valid entry in the trace buffer. The msb of this field can be used to determine if the entire PST trace buffer has been loaded with valid data. <table border="1" style="margin-left: 20px;"> <tr> <th>PSTBWA[7]</th> <th>PSTB Valid Data Locations (Oldest to Newest)</th> </tr> <tr> <td>0</td> <td>0, 1, ... PSTBWA-1</td> </tr> <tr> <td>1</td> <td>PSTBWA, PSTBWA+1,..., 0, 1, PSTBWA-1</td> </tr> </table> <p>The PSTBWA is unaffected when a buffer stop condition has been reached, the buffer is disabled, or a system reset occurs. This allows the contents of the PST trace buffer to be retrieved after these events to assist in debug. <b>Note:</b> Since this device contains a 64-entry trace buffer, PSTBWA[6] is always zero.</p>	PSTBWA[7]	PSTB Valid Data Locations (Oldest to Newest)	0	0, 1, ... PSTBWA-1	1	PSTBWA, PSTBWA+1,..., 0, 1, PSTBWA-1
PSTBWA[7]	PSTB Valid Data Locations (Oldest to Newest)						
0	0, 1, ... PSTBWA-1						
1	PSTBWA, PSTBWA+1,..., 0, 1, PSTBWA-1						
7 PSTBR	PST trace buffer reset. Generates a reset of the PST trace buffer logic, which clears PSTBWA and PSTBST. The same resources are reset when a disabled trace buffer becomes enabled and upon the receipt of a BDM GO command when operating in obtrusive trace mode. These reset events also clear any accumulation of PSTs. This bit always reads as a zero. 0 Do not force a PST trace buffer reset 1 Force a PST trace buffer reset						

**Table 20-9. CSR2 Field Descriptions (continued)**

Field	Description																								
6 APCDIV16	Automatic PC synchronization divide cycle counts by 16. This bit divides the cycle counts for automatic SYNC_PC command insertion by 16. See the APCSC and APCENB field descriptions.																								
5	Reserved, must be cleared.																								
4–3 PSTBRM	PST trace buffer recording mode. Defines the trace buffer recording mode. The start and stop recording conditions are defined by the PSTBSS field. 00 Non-obtrusive, normal recording mode 01 Obtrusive, normal recording 10 Non-obtrusive, PC profile recording. Automatic PC synchronization must be enabled (see XCSR[APCSC, APCENB], CSR2[APCDIV16], and CSR[BTB]). 11 Obtrusive, PC profile recording. Automatic PC synchronization must be enabled (see XCSR[APCSC, APCENB], CSR2[APCDIV16], and CSR[BTB]). The terms obtrusive and non-obtrusive are defined as: <ul style="list-style-type: none"> <li>• Non-obtrusive—The core is not halted. The PST trace buffer is overwritten unless a PSTB start/stop combination results in less than or equal to 64 PSTB captures.</li> <li>• Obtrusive—The core is halted when the PSTB trace buffer reaches its full level (full before overwriting). The PSTB trace buffer contents are available by the BDM PSTB_READ commands. The PSTB trace buffer write address resets and the CPU resumes upon a BDM GO command.</li> </ul>																								
2–0 PSTBSS	PST trace buffer start/stop definition. Specifies the start and stop conditions for PST trace buffer recording. In certain cases, the start and stop conditions are defined by the breakpoint registers. The remaining breakpoint registers are available for trigger configurations. <table border="1" style="margin-left: 20px;"> <thead> <tr> <th>PSTBSS</th> <th>Start Condition</th> <th>Stop Condition</th> </tr> </thead> <tbody> <tr> <td>000</td> <td colspan="2">Trace buffer disabled, no recording</td> </tr> <tr> <td>001</td> <td colspan="2">Unconditional recording</td> </tr> <tr> <td>010</td> <td rowspan="2">ABxR{&amp; DBR/DBMR}</td> <td>PBR0/PBMR</td> </tr> <tr> <td>011</td> <td>PBR1</td> </tr> <tr> <td>100</td> <td rowspan="2">PBR0/PBMR</td> <td>ABxR{&amp; DBR/DBMR}</td> </tr> <tr> <td>101</td> <td>PBR1</td> </tr> <tr> <td>110</td> <td rowspan="2">PBR1</td> <td>ABxR{&amp; DBR/DBMR}</td> </tr> <tr> <td>111</td> <td>PBR0/PBMR</td> </tr> </tbody> </table>	PSTBSS	Start Condition	Stop Condition	000	Trace buffer disabled, no recording		001	Unconditional recording		010	ABxR{& DBR/DBMR}	PBR0/PBMR	011	PBR1	100	PBR0/PBMR	ABxR{& DBR/DBMR}	101	PBR1	110	PBR1	ABxR{& DBR/DBMR}	111	PBR0/PBMR
PSTBSS	Start Condition	Stop Condition																							
000	Trace buffer disabled, no recording																								
001	Unconditional recording																								
010	ABxR{& DBR/DBMR}	PBR0/PBMR																							
011		PBR1																							
100	PBR0/PBMR	ABxR{& DBR/DBMR}																							
101		PBR1																							
110	PBR1	ABxR{& DBR/DBMR}																							
111		PBR0/PBMR																							

### 20.3.4 Configuration/Status Register 3 (CSR3)

CSR3 contains the BDM flash clock divider (BFCDIV) value in a format similar to HCS08 devices.

There are multiple ways to reference CSR3. They are summarized in [Table 20-10](#).

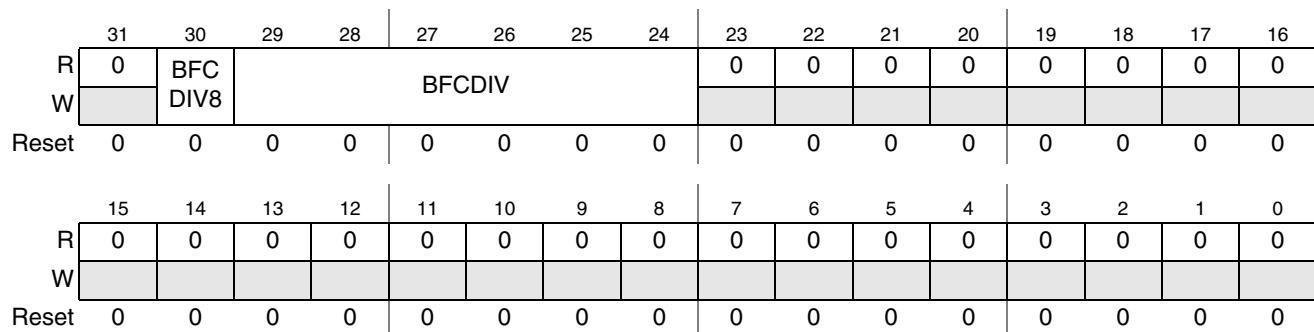
**Table 20-10. CSR3 Reference Summary**

Method	Reference Details
READ_CSR3_BYTE	Reads CSR3[31–24] from the BDM interface. Available in all modes.
WRITE_CSR3_BYTE	Writes CSR3[31–24] from the BDM interface. Available in all modes.

**Table 20-10. CSR3 Reference Summary (continued)**

Method	Reference Details
READ_DREG	Reads CSR3[31–0] from the BDM interface. Classified as a non-intrusive BDM command.
WRITE_DREG	Writes CSR3[23–0] from the BDM interface. Classified as a non-intrusive BDM command.
WDEBUG Instruction	No operation during the core's execution of a WDEBUG instruction

DRc: 0x03 (CSR3)

Access: Supervisor write-only  
BDM read/write**Figure 20-6. Configuration/Status Register 3 (CSR3)****Table 20-11. CSR3 Field Descriptions**

Field	Description
31	Reserved, must be cleared.
30 BFCDIV8	BDM flash clock divide by 8. 0 Input to the flash clock divider is the bus clock 1 Input to the flash clock divider is the bus clock divided by 8
29–24 BFCDIV	BDM flash clock divider. The BFCDIV8 and BFCDIV fields specify the frequency of the internal flash clock when performing a mass erase operation initiated by setting XCSR[ERASE]. These fields must be loaded with the appropriate values prior to the setting of XCSR[ERASE] to initiate a mass erase operation in the flash memory.  This field divides the bus clock (or the bus clock divided by 8 if BFCDIV8 is set) by the value defined by the BFCDIV plus one. The resulting frequency of the internal flash clock must fall within the range of 150–200 kHz for proper flash operations. Program/erase timing pulses are one cycle of this internal flash clock, which corresponds to a range of 5–6.7 µs. The automated programming logic uses an integer number of these pulses to complete an erase or program operation.  if BFCDIV8 = 0, then $f_{FCLK} = f_{Bus} \div (BFCDIV + 1)$ if BFCDIV8 = 1, then $f_{FCLK} = f_{Bus} \div (8 \times (BFCDIV + 1))$  where $f_{FCLK}$ is the frequency of the flash clock and $f_{Bus}$ is the frequency of the bus clock.
23–0	Reserved for future use by the debug module, must be cleared.

### 20.3.5 BDM Address Attribute Register (BAAR)

BAAR defines the address space for memory-referencing BDM commands. BAAR[R, SZ] are loaded directly from the BDM command, while the lower five bits can be programmed from the external

development system. BAAR is loaded any time AATR is written and is initialized to a value of 0x05, setting supervisor data as the default address space. The upper 24 bits of this register are reserved for future use and any attempted write of these bits is ignored.

DRc: 0x05 (BAAR)

Access: Supervisor write-only  
BDM write-only

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

Figure 20-7. BDM Address Attribute Register (BAAR)

Table 20-12. BAAR Field Descriptions

Field	Description
31–8	Reserved for future use by the debug module, must be cleared.
7 R	Read/Write. 0 Write 1 Read
6–5 SZ	Size. 00 Longword 01 Byte 10 Word 11 Reserved
4–3 TT	Transfer type. See the TT definition in the AATR description, <a href="#">Section 20.3.6, “Address Attribute Trigger Register (AATR)”</a> .
2–0 TM	Transfer modifier. See the TM definition in the AATR description, <a href="#">Section 20.3.6, “Address Attribute Trigger Register (AATR)”</a> .

### 20.3.6 Address Attribute Trigger Register (AATR)

AATR 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 high-speed local 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 WRITE\_DREG command.

DRc: 0x06 (AATR)

Access: Supervisor write-only  
BDM write-only

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R																																
W	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	RM	SZM	TTM	TMM	R	SZ	TT	TM	0	0	0	0	0	0	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

Figure 20-8. Address Attribute Trigger Register (AATR)

**Table 20-13. AATR Field Descriptions**

Field	Description
31–16	Reserved, must be cleared.
15 RM	Read/write mask. Masks the R bit in address comparisons.
14–13 SZM	Size mask. Masks the corresponding SZ bit in address comparisons.
12–11 TTM	Transfer type mask. Masks the corresponding TT bit in address comparisons.
10–8 TMM	Transfer modifier mask. 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. These bits also define the TT encoding for BDM memory commands. 00 Normal processor access Else Reserved
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). 000 Reserved 001 User-mode data access 010 User-mode code access 011 Reserved 100 Reserved 101 Supervisor-mode data access 110 Supervisor-mode code access 111 Reserved

### 20.3.7 Trigger Definition Register (TDR)

TDR configures the operation of the hardware breakpoint logic that corresponds with the ABHR/ABLR/AATR, PBR/PBR1/PBR2/PBR3/PBMR, and DBR/DBMR registers within the debug module. TDR controls the actions taken under the defined conditions. Breakpoint logic may be configured as one- or two-level trigger. TDR[31–16] defines the second-level trigger, and TDR[15–0] defines the first-level trigger.

#### NOTE

The debug module has no hardware interlocks. To prevent spurious breakpoint triggers while the breakpoint registers are being loaded, disable TDR (clear TDR[L2EBL,L1EBL]) 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 WRITE\_DREG command.

DRc: 0x07 (TDR)

Access: Supervisor write-only  
BDM write-only

Second Level Trigger																
R	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
W	TRC	L2EBL	L2ED						L2DI	L2EA			L2EPC	L2PCI		
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
First Level Trigger																
R	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
W	L2T	L1T	L1EBL	L1ED						L1DI	L1EA			L1EPC	L1PCI	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 20-9. Trigger Definition Register (TDR)

Table 20-14. TDR Field Descriptions

Field	Description																
31–30 TRC	Trigger response control. Determines how the processor responds to a completed trigger condition. The trigger response is displayed on PST. 00 Display on PST only 01 Processor halt 10 Debug interrupt 11 Reserved																
29 L2EBL	Enable level 2 breakpoint. Global enable for the breakpoint trigger. 0 Disables all level 2 breakpoints 1 Enables all level 2 breakpoint triggers																
28–22 L2ED	Enable level 2 data breakpoint. Setting an L2ED bit enables the corresponding data breakpoint condition based on the size and placement on the processor's local data bus. Clearing all ED bits disables data breakpoints.																
	<table border="1"> <thead> <tr> <th>TDR Bit</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>28</td> <td>Data longword. Entire processor's local data bus.</td></tr> <tr> <td>27</td> <td>Lower data word.</td></tr> <tr> <td>26</td> <td>Upper data word.</td></tr> <tr> <td>25</td> <td>Lower lower data byte. Low-order byte of the low-order word.</td></tr> <tr> <td>24</td> <td>Lower middle data byte. High-order byte of the low-order word.</td></tr> <tr> <td>23</td> <td>Upper middle data byte. Low-order byte of the high-order word.</td></tr> <tr> <td>22</td> <td>Upper upper data byte. High-order byte of the high-order word.</td></tr> </tbody> </table>	TDR Bit	Description	28	Data longword. Entire processor's local data bus.	27	Lower data word.	26	Upper data word.	25	Lower lower data byte. Low-order byte of the low-order word.	24	Lower middle data byte. High-order byte of the low-order word.	23	Upper middle data byte. Low-order byte of the high-order word.	22	Upper upper data byte. High-order byte of the high-order word.
TDR Bit	Description																
28	Data longword. Entire processor's local data bus.																
27	Lower data word.																
26	Upper data word.																
25	Lower lower data byte. Low-order byte of the low-order word.																
24	Lower middle data byte. High-order byte of the low-order word.																
23	Upper middle data byte. Low-order byte of the high-order word.																
22	Upper upper data byte. High-order byte of the high-order word.																

**Table 20-14. TDR Field Descriptions (continued)**

<b>Field</b>	<b>Description</b>									
21 L2DI	Level 2 data breakpoint invert. Inverts 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. 0 No inversion 1 Invert data breakpoint comparators.									
20–18 L2EA	Enable level 2 address breakpoint. Setting an L2EA bit enables the corresponding address breakpoint. Clearing all three bits disables the breakpoint. <table border="1" data-bbox="479 502 1258 792"> <thead> <tr> <th><b>TDR Bit</b></th><th><b>Description</b></th></tr> </thead> <tbody> <tr> <td>20</td><td>Address breakpoint inverted. Breakpoint is based outside the range between ABLR and ABHR.</td></tr> <tr> <td>19</td><td>Address breakpoint range. The breakpoint is based on the inclusive range defined by ABLR and ABHR.</td></tr> <tr> <td>18</td><td>Address breakpoint low. The breakpoint is based on the address in the ABLR.</td></tr> </tbody> </table>		<b>TDR Bit</b>	<b>Description</b>	20	Address breakpoint inverted. Breakpoint is based outside the range between ABLR and ABHR.	19	Address breakpoint range. The breakpoint is based on the inclusive range defined by ABLR and ABHR.	18	Address breakpoint low. The breakpoint is based on the address in the ABLR.
<b>TDR Bit</b>	<b>Description</b>									
20	Address breakpoint inverted. Breakpoint is based outside the range between ABLR and ABHR.									
19	Address breakpoint range. The breakpoint is based on the inclusive range defined by ABLR and ABHR.									
18	Address breakpoint low. The breakpoint is based on the address in the ABLR.									
17 L2EPC	Enable level 2 PC breakpoint. 0 Disable PC breakpoint 1 Enable PC breakpoint									
16 L2PCI	Level 2 PC breakpoint invert. 0 The PC breakpoint is defined within the region defined by PBR <sub>n</sub> and PBMR. 1 The PC breakpoint is defined outside the region defined by PBR <sub>n</sub> and PBMR.									
15 L2T	Level 2 trigger. Determines the logic operation for the trigger between the PC_condition and the (Address_range and Data) condition where the inclusion of a Data_condition is optional. The ColdFire debug architecture supports the creation of single or double-level triggers. 0 Level 2 trigger = PC_condition && (Address_range && Data_condition) 1 Level 2 trigger = PC_condition    (Address_range && Data_condition)									
14 L1T	Level 1 trigger. Determines the logic operation for the trigger between the PC_condition and the (Address_range and Data) condition where the inclusion of a Data_condition is optional. The ColdFire debug architecture supports the creation of single or double-level triggers. 0 Level 1 trigger = PC_condition && (Address_range && Data_condition) 1 Level 1 trigger = PC_condition    (Address_range && Data_condition)									
13 L1EBL	Enable level 1 breakpoint. Global enable for the breakpoint trigger. 0 Disables all level 1 breakpoints 1 Enables all level 1 breakpoint triggers									

**Table 20-14. TDR Field Descriptions (continued)**

<b>Field</b>	<b>Description</b>																	
12–6 L1ED	Enable level 1 data breakpoint. Setting an L1ED bit enables the corresponding data breakpoint condition based on the size and placement on the processor's local data bus. Clearing all L1ED bits disables data breakpoints.																	
	<table border="1"> <thead> <tr> <th><b>TDR Bit</b></th><th><b>Description</b></th></tr> </thead> <tbody> <tr> <td>12</td><td>Data longword. Entire processor's local data bus.</td></tr> <tr> <td>11</td><td>Lower data word.</td></tr> <tr> <td>10</td><td>Upper data word.</td></tr> <tr> <td>9</td><td>Lower lower data byte. Low-order byte of the low-order word.</td></tr> <tr> <td>8</td><td>Lower middle data byte. High-order byte of the low-order word.</td></tr> <tr> <td>7</td><td>Upper middle data byte. Low-order byte of the high-order word.</td></tr> <tr> <td>6</td><td>Upper upper data byte. High-order byte of the high-order word.</td></tr> </tbody> </table>		<b>TDR Bit</b>	<b>Description</b>	12	Data longword. Entire processor's local data bus.	11	Lower data word.	10	Upper data word.	9	Lower lower data byte. Low-order byte of the low-order word.	8	Lower middle data byte. High-order byte of the low-order word.	7	Upper middle data byte. Low-order byte of the high-order word.	6	Upper upper data byte. High-order byte of the high-order word.
<b>TDR Bit</b>	<b>Description</b>																	
12	Data longword. Entire processor's local data bus.																	
11	Lower data word.																	
10	Upper data word.																	
9	Lower lower data byte. Low-order byte of the low-order word.																	
8	Lower middle data byte. High-order byte of the low-order word.																	
7	Upper middle data byte. Low-order byte of the high-order word.																	
6	Upper upper data byte. High-order byte of the high-order word.																	
5 L1DI	Level 1 data breakpoint invert. Inverts 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. 0 No inversion 1 Invert data breakpoint comparators.																	
4–2 L1EA	Enable level 1 address breakpoint. Setting an L1EA bit enables the corresponding address breakpoint. Clearing all three bits disables the address breakpoint.																	
	<table border="1"> <thead> <tr> <th><b>TDR Bit</b></th><th><b>Description</b></th></tr> </thead> <tbody> <tr> <td>4</td><td>Enable address breakpoint inverted. Breakpoint is based outside the range between ABLR and ABHR.</td></tr> <tr> <td>3</td><td>Enable address breakpoint range. The breakpoint is based on the inclusive range defined by ABLR and ABHR.</td></tr> <tr> <td>2</td><td>Enable address breakpoint low. The breakpoint is based on the address in the ABLR.</td></tr> </tbody> </table>		<b>TDR Bit</b>	<b>Description</b>	4	Enable address breakpoint inverted. Breakpoint is based outside the range between ABLR and ABHR.	3	Enable address breakpoint range. The breakpoint is based on the inclusive range defined by ABLR and ABHR.	2	Enable address breakpoint low. The breakpoint is based on the address in the ABLR.								
<b>TDR Bit</b>	<b>Description</b>																	
4	Enable address breakpoint inverted. Breakpoint is based outside the range between ABLR and ABHR.																	
3	Enable address breakpoint range. The breakpoint is based on the inclusive range defined by ABLR and ABHR.																	
2	Enable address breakpoint low. The breakpoint is based on the address in the ABLR.																	
1 L1EPC	Enable level 1 PC breakpoint. 0 Disable PC breakpoint 1 Enable PC breakpoint																	
0 L1PCI	Level 1 PC breakpoint invert. 0 The PC breakpoint is defined within the region defined by PBR $n$ and PBMR. 1 The PC breakpoint is defined outside the region defined by PBR $n$ and PBMR.																	

### 20.3.8 Program Counter Breakpoint/Mask Registers (PBR0–3, PBMR)

The PBR $n$  registers define instruction addresses for use as part of the trigger. These registers' contents are compared with the processor's program counter register when the appropriate valid bit is set (for PBR1–3) and TDR is configured appropriately. PBR0 bits are masked by setting corresponding PBMR bits (PBMR has no effect on PBR1–3). Results are compared with the processor's program counter register, as defined in TDR. The PC breakpoint registers, PBR1–3, have no masking associated with them, but do include a

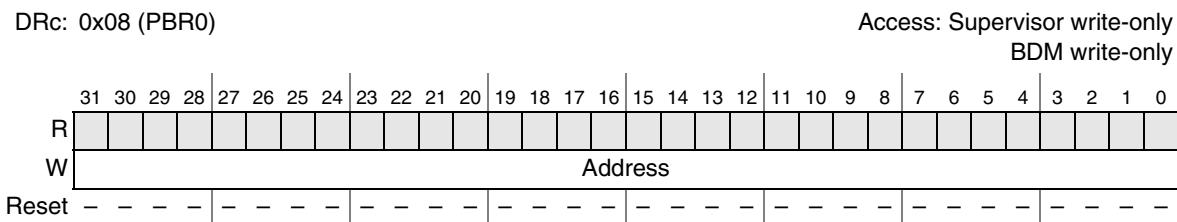
valid bit. These registers' contents are compared with the processor's program counter register when TDR is configured appropriately.

The PC breakpoint registers are accessible in supervisor mode using the WDEBUG instruction and through the BDM port using the WRITE\_DREG command using values shown in [Section 20.4.1.4, “BDM Command Set Descriptions”](#).

### NOTE

Version 1 ColdFire core devices implement a 24-bit, 16 MB address map.

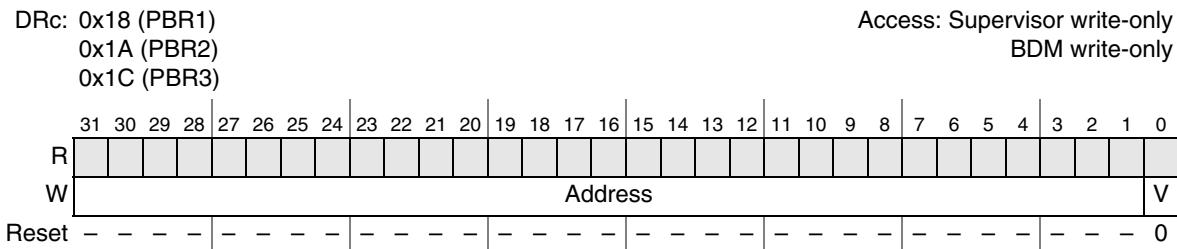
When programming these registers with a 32-bit address, the upper byte must be zero-filled.



**Figure 20-10. Program Counter Breakpoint Register 0 (PBR0)**

**Table 20-15. PBR0 Field Descriptions**

Field	Description
31–0 Address	PC breakpoint address. The address to be compared with the PC as a breakpoint trigger. Because all instruction sizes are multiples of 2 bytes, bit 0 of the address should always be zero.



**Figure 20-11. Program Counter Breakpoint Register  $n$  (PBR $n$ ,  $n = 1,2,3$ )**

**Table 20-16. PBR $n$  Field Descriptions**

Field	Description
31–1 Address	PC breakpoint address. The 31-bit address to be compared with the PC as a breakpoint trigger.
0 V	Valid bit. This bit must be set for the PC breakpoint to occur at the address specified in the Address field. 0 PBR is disabled. 1 PBR is enabled.

[Figure 20-12](#) shows PBMR. PBMR is accessible in supervisor mode using the WDEBUG instruction and via the BDM port using the WRITE\_DREG command. PBMR only masks PBR0.

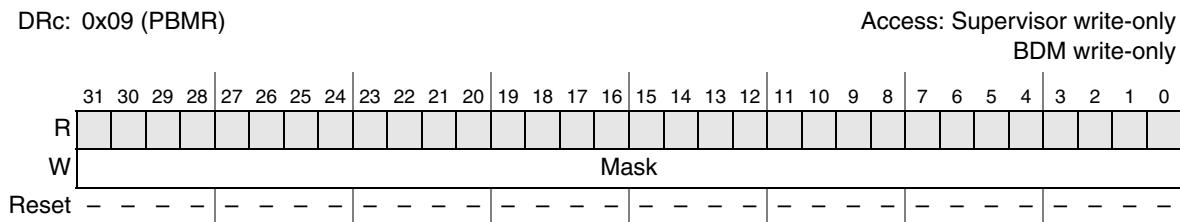


Figure 20-12. Program Counter Breakpoint Mask Register (PBMR)

Table 20-17. PBMR Field Descriptions

Field	Description
31–0 Mask	PC breakpoint mask. If using PBR0, this register must be initialized since it is undefined after reset. 0 The corresponding PBR0 bit is compared to the appropriate PC bit. 1 The corresponding PBR0 bit is ignored.

### 20.3.9 Address Breakpoint Registers (ABLR, ABHR)

The ABLR and ABHR 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:

- Identical to the value in ABLR
- Inside the range bound by ABLR and ABHR inclusive
- Outside that same range

The address breakpoint registers are accessible in supervisor mode using the WDEBUG instruction and through the BDM port using the WRITE\_DREG command using values shown in [Section 20.4.1.4, “BDM Command Set Descriptions.”](#)

#### NOTE

Version 1 ColdFire core devices implement a 24-bit, 16 MB address map. When programming these registers with a 32-bit address, the upper byte must be zero-filled.

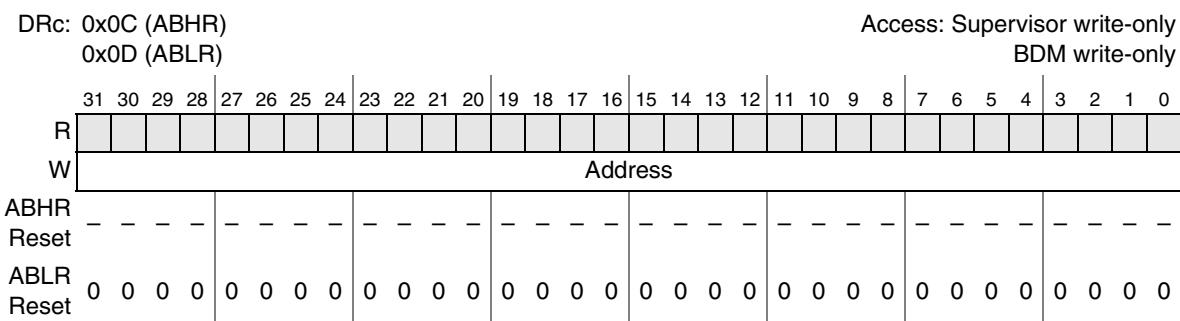


Figure 20-13. Address Breakpoint Registers (ABLR, ABHR)

**Table 20-18. ABLR Field Description**

Field	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.

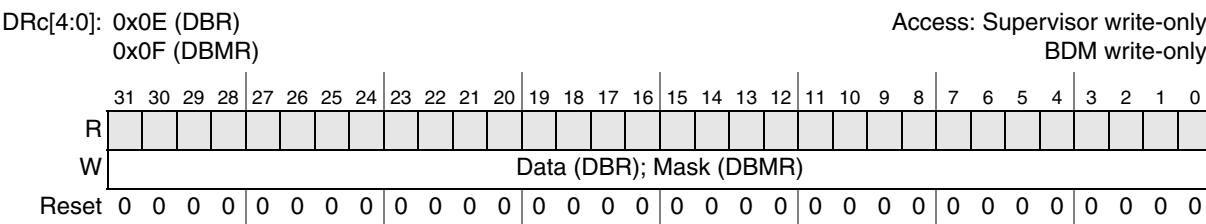
**Table 20-19. ABHR Field Description**

Field	Description
31–0 Address	High address. Holds the 32-bit address marking the upper bound of the address breakpoint range.

### 20.3.10 Data Breakpoint and Mask Registers (DBR, DBMR)

DBR 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 and DBMR are accessible in supervisor mode using the WDEBUG instruction and through the BDM port using the WRITE\_DREG commands.

**Figure 20-14. Data Breakpoint & Mask Registers (DBR, DBMR)****Table 20-20. DBR Field Descriptions**

Field	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 20-21. DBMR Field Descriptions**

Field	Description
31–0 Mask	Data breakpoint mask. The 32-bit mask for the data breakpoint trigger. 0 The corresponding DBR bit is compared to the appropriate bit of the processor's local data bus 1 The corresponding DBR bit is ignored

The DBR supports aligned and misaligned references. [Table 20-22](#) shows the relationships between processor address, access size, and location within the 32-bit data bus.

**Table 20-22. Access Size and Operand Data Location**

Address[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]

### 20.3.10.1 Resulting Set of Possible Trigger Combinations

The resulting set of possible breakpoint trigger combinations consists of the following options where || denotes logical OR, && denotes logical AND, and {} denotes an optional additional trigger term:

One-level triggers of the form:

```
if      (PC_breakpoint)
if      (PC_breakpoint || Address_breakpoint{&& Data_breakpoint})
if      (Address_breakpoint {&& Data_breakpoint})
```

Two-level triggers of the form:

```
if      (PC_breakpoint)
      then if (Address_breakpoint{&& Data_breakpoint})

if      (Address_breakpoint {&& Data_breakpoint})
      then if (PC_breakpoint)
```

In these examples, PC\_breakpoint is the logical summation of the PBR0/PBMR, PBR1, PBR2, and PBR3 breakpoint registers; Address\_breakpoint is a function of ABHR, ABLR, and AATR; Data\_breakpoint is a function of DBR and DBMR. In all cases, the data breakpoints can be included with an address breakpoint to further qualify a trigger event as an option.

The breakpoint registers can also be used to define the start and stop recording conditions for the PST trace buffer. For information on this functionality, see [Section 20.3.3, “Configuration/Status Register 2 \(CSR2\)”](#).

### 20.3.11 PST Buffer (PSTB)

The PST trace buffer contains 64 six-bit entries, packed consecutively into 12 longword locations. See [Figure 20-15](#) for an illustration of how the buffer entries are packed.

The write pointer for the trace buffer is available as CSR2[PSTBWA]. Using this pointer, it is possible to determine the oldest-to-newest entries in the trace buffer.

Core register number (CRN)	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x10	TB #00				TB #01				TB #02				TB #03				TB #04				05[5:4]											
0x11	TB #05[3:0]				TB #06				TB #07				TB #08				TB #09				TB #10[5:2]											
0x12	10[1:0]	TB #11				TB #12				TB #13				TB #14				TB #15														
0x13	TB #16				TB #17				TB #18				TB #19				TB #20				21[5:4]											
0x14	TB #21[3:0]				TB #22				TB #23				TB #24				TB #25				TB #26[5:2]											
0x15	26[1:0]	TB #27				TB #28				TB #29				TB #30				TB #31														
0x16	TB #32				TB #33				TB #34				TB #35				TB #36				37[5:4]											
0x17	TB #37[3:0]				TB #38				TB #39				TB #40				TB #41				TB #42[5:2]											
0x18	42[1:0]	TB #43				TB #44				TB #45				TB #46				TB #47														
0x19	TB #48				TB #49				TB #50				TB #51				TB #52				53[5:4]											
0x1A	TB #53[3:0]				TB #54				TB #55				TB #56				TB #57				TB #58[5:2]											
0x1B	58[1:0]	TB #59				TB #60				TB #61				TB #62				TB #63														

Figure 20-15. PST Trace Buffer Entries and Locations

## 20.4 Functional Description

### 20.4.1 Background Debug Mode (BDM)

This section provides details on the background debug serial interface controller (BDC) and the BDM command set.

The BDC provides a single-wire debug interface to the target MCU. As shown in the Version 1 ColdFire core block diagram of [Figure 20-1](#), the BDC module interfaces between the single-pin (BKGD) interface and the remaining debug modules, including the ColdFire background debug logic, the real-time debug hardware, and the PST/DDATA trace logic. This interface provides a convenient means for programming the on-chip flash and other non-volatile memories. The BDC is the primary debug interface for development and allows non-intrusive access to memory data and traditional debug features such as run/halt control, read/write of core registers, breakpoints, and single instruction step.

Features of the background debug controller (BDC) include:

- Single dedicated pin for mode selection and background communications
- Special BDC registers not located in system memory map
- SYNC command to determine target communications rate
- Non-intrusive commands for memory access
- Active background (halt) mode commands for core register access
- GO command to resume execution
- BACKGROUND command to halt core or wake CPU from low-power modes
- Oscillator runs in stop mode, if BDM enabled

Based on these features, BDM is useful for the following reasons:

- In-circuit emulation is not needed, so physical and electrical characteristics of the system are not affected.
- BDM is always available for debugging the system and provides a communication link for upgrading firmware in existing systems.
- Provides high-speed memory downloading, especially useful for flash programming
- Provides absolute control of the processor, and thus the system. This feature allows quick hardware debugging with the same tool set used for firmware development.

#### 20.4.1.1 CPU Halt

Although certain 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. Recall that the default configuration of the Version 1 ColdFire core (CF1Core) defines the occurrence of certain exception types to automatically generate a system reset. Some of these fault types include illegal instructions, privilege errors, address errors, and bus error terminations, with the response under control of the processor's CPUCR[ARD,IRD] bits.

**Table 20-23. CPU Halt Sources**

Halt Source	Halt Timing	Description		
Fault-on-fault	Immediate	Refers to the occurrence of any fault while exception processing. For example, a bus error is signaled during exception stack frame writes or while fetching the first instruction in the exception service routine.		
		CPUCR[ARD] = 1   Immediately enters halt.		
		CPUCR[ARD] = 0   Reset event is initiated.		
Hardware breakpoint trigger	Pending	Halt is made pending in the processor. 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.		
HALT instruction	Immediate	BDM disabled	CPUCR[IRD] = 0	A reset is initiated since attempted execution of an illegal instruction
			CPUCR[IRD] = 1	An illegal instruction exception is generated.
		BDM enabled, supervisor mode	Processor immediately halts execution at the next instruction sample point. The processor can be restarted by a BDM GO command. Execution continues at the instruction after HALT.	
		BDM enabled, user mode	CSR[UHE] = 0 CPUCR[IRD] = 0	A reset event is initiated, because a privileged instruction was attempted in user mode.
			CSR[UHE] = 0 CPUCR[IRD] = 1	A privilege violation exception is generated.
			CSR[UHE] = 1	Processor immediately halts execution at the next instruction sample point. The processor can be restarted by a BDM GO command. Execution continues at the instruction after HALT.

**Table 20-23. CPU Halt Sources (continued)**

Halt Source	Halt Timing	Description		
BACKGROUND command	Pending	BDM disabled or flash secure	Illegal command response and BACKGROUND command is ignored.	
		BDM enabled and flash unsecure	Processor is running	Halt is made pending in the processor. 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.
			Processor is stopped	Processing of the BACKGROUND command is treated in a special manner. 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 (the instruction following STOP).
PSTB full condition	Pending	PSTB	PSTB obtrusive recording mode pends halt in the processor if the trace buffer reaches its full threshold (full is defined as before the buffer is overwritten). When a pending condition is asserted, the processor halts at the next sample point.	
BKGD held low for $\geq 2$ bus clocks after reset negated for POR or BDM reset	Immediate	Flash unsecure	Enters debug mode with XCSR[ENBDM, CLKSW] set. The full set of BDM commands is available and debug can proceed. If the core is reset into a debug halt condition, the processor's response to the GO command depends on the BDM command(s) performed while it was halted. 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.	
		Flash secure	Enters debug mode with XCSR[ENBDM, CLKSW] set. The allowable commands are limited to the always-available group. A GO command to start the processor is not allowed. The only recovery actions in this mode are: <ul style="list-style-type: none"> <li>• Issue a BDM reset setting CSR2[BDFR] with CSR2[BDHBR] cleared and the BKGD pin held high to reset into normal operating mode</li> <li>• Erase the flash to unsecure the memory and then proceed with debug</li> <li>• Power cycle the device with the BKGD pin held high to reset into the normal operating mode</li> </ul>	

The processor's run/stop/halt status is always accessible in XCSR[CPUHALT,CPUSTOP]. Additionally, CSR[27–24] indicate the halt source, showing the highest priority source for multiple halt conditions. This field is cleared by a read of the CSR. A processor halt due to the PSTB full condition as indicated by CSR2[PSTH] is also reflected in CSR[BKPT]. The debug GO command clears CSR[26–24] and CSR2[PSTBH].

#### 20.4.1.2 Background Debug Serial Interface Controller (BDC)

BDC serial communications use a custom serial protocol first introduced on the M68HC12 Family of microcontrollers and later used in the M68HCS08 family. This protocol assumes that the host knows the

communication clock rate determined by the target BDC clock rate. The BDC clock rate may be the system bus clock frequency or an alternate frequency source depending on the state of XCSR[CLKSW]. All communication is initiated and controlled by the host which drives a high-to-low edge to signal the beginning of each bit time. Commands and data are sent most significant bit (msb) first. For a detailed description of the communications protocol, refer to [Section 20.4.1.3, “BDM Communication Details”](#).

If a host is attempting to communicate with a target MCU that has an unknown BDC clock rate, a SYNC command may be sent to the target MCU to request a timed synchronization response signal from which the host can determine the correct communication speed. After establishing communications, the host can read XCSR and write the clock switch (CLKSW) control bit to change the source of the BDC clock for further serial communications if necessary.

BKGD is a pseudo-open-drain pin and there is an on-chip pullup so no external pullup resistor is required. Unlike typical open-drain pins, the external RC time constant on this pin, which is influenced by external capacitance, plays almost no role in signal rise time. The custom protocol provides for brief, actively driven speed-up pulses to force rapid rise times on this pin without risking harmful drive level conflicts. Refer to [Section 20.4.1.3, “BDM Communication Details,”](#) for more details.

When no debugger pod is connected to the standard 6-pin BDM interface connector ([Section 20.4.4, “Freescale-Recommended BDM Pinout”](#)), the internal pullup on BKGD chooses normal operating mode. When a development system is connected, it can pull BKGD and  $\overline{\text{RESET}}$  low, release  $\overline{\text{RESET}}$  to select active background (halt) mode rather than normal operating mode, and then release BKGD. It is not necessary to reset the target MCU to communicate with it through the background debug interface. There is also a mechanism to generate a reset event in response to setting CSR2[BDFR].

### **20.4.1.3 BDM Communication Details**

The BDC serial interface requires the external host controller to generate a falling edge on the BKGD pin to indicate the start of each bit time. The external controller provides this falling edge whether data is transmitted or received.

BKGD is a pseudo-open-drain pin that can be driven by an external controller or by the MCU. Data is transferred msb first at 16 BDC clock cycles per bit (nominal speed). The interface times-out if 512 BDC clock cycles occur between falling edges from the host. If a time-out occurs, the status of any command in progress must be determined before new commands can be sent from the host. To check the status of the command, follow the steps detailed in the bit description of XCSR[CSTAT] in [Table 20-7](#).

The custom serial protocol requires the debug pod to know the target BDC communication clock speed. The clock switch (CLKSW) control bit in the XCSR[31–24] register allows you to select the BDC clock source. The BDC clock source can be the bus clock or the alternate BDC clock source. When the MCU is reset in normal user mode, CLKSW is cleared and that selects the alternate clock source. This clock source is a fixed frequency independent of the bus frequency so it does change if the user modifies clock generator settings. This is the preferred clock source for general debugging.

When the MCU is reset in active background (halt) mode, CLKSW is set which selects the bus clock as the source of the BDC clock. This CLKSW setting is most commonly used during flash memory programming because the bus clock can usually be configured to operate at the highest allowed bus frequency to ensure the fastest possible flash programming times. Because the host system is in control of

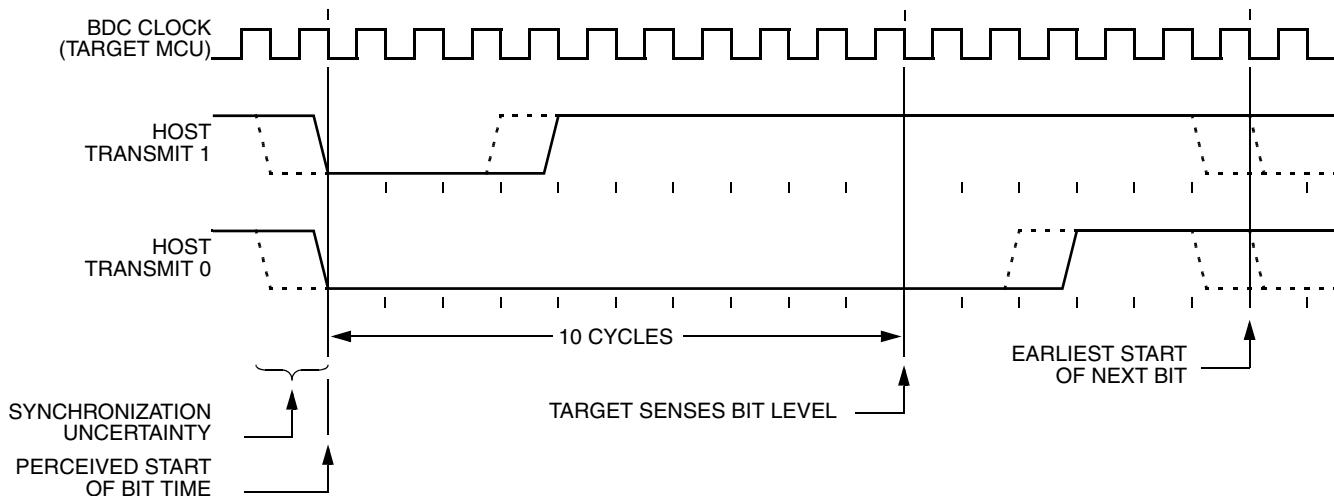
changes to clock generator settings, it knows when a different BDC communication speed should be used. The host programmer also knows that no unexpected change in bus frequency could occur to disrupt BDC communications.

Normally, setting CLKSW should not be used for general debugging because there is no way to ensure the application program does not change the clock generator settings. This is especially true in the case of application programs that are not yet fully debugged.

After any reset (or at any other time), the host system can issue a SYNC command to determine the speed of the BDC clock. CLKSW may be written using the serial WRITE\_XCSR\_BYT command through the BDC interface. CLKSW is located in the special XCSR byte register in the BDC module and it is not accessible in the normal memory map of the ColdFire core. This means that no program running on the processor can modify this register (intentionally or unintentionally).

The BKGD pin can receive a high- or low-level or transmit a high- or low-level. The following diagrams show timing for each of these cases. Interface timing is synchronous to clocks in the target BDC, but asynchronous to the external host. The internal BDC clock signal is shown for reference in counting cycles.

**Figure 20-16** shows an external host transmitting a logic 1 or 0 to the BKGD pin of a target MCU. The host is asynchronous to the target so there is a 0–1 cycle delay from the host-generated falling edge to where the target perceives the beginning of the bit time. Ten target BDC clock cycles later, the target senses the bit level on the BKGD pin. Typically, the host actively drives the pseudo-open-drain BKGD pin during host-to-target transmissions to speed up rising edges. Because the target does not drive the BKGD pin during the host-to-target transmission period, there is no need to treat the line as an open-drain signal during this period.



**Figure 20-16. BDC Host-to-Target Serial Bit Timing**

**Figure 20-17** shows the host receiving a logic 1 from the target MCU. Because the host is asynchronous to the target MCU, there is a 0–1 cycle delay from the host-generated falling edge on BKGD to the perceived start of the bit time in the target MCU. The host holds the BKGD pin low long enough for the target to recognize it (at least two target BDC cycles). The host must release the low drive before the target

MCU drives a brief active-high speedup pulse seven cycles after the perceived start of the bit time. The host should sample the bit level about 10 cycles after it started the bit time.

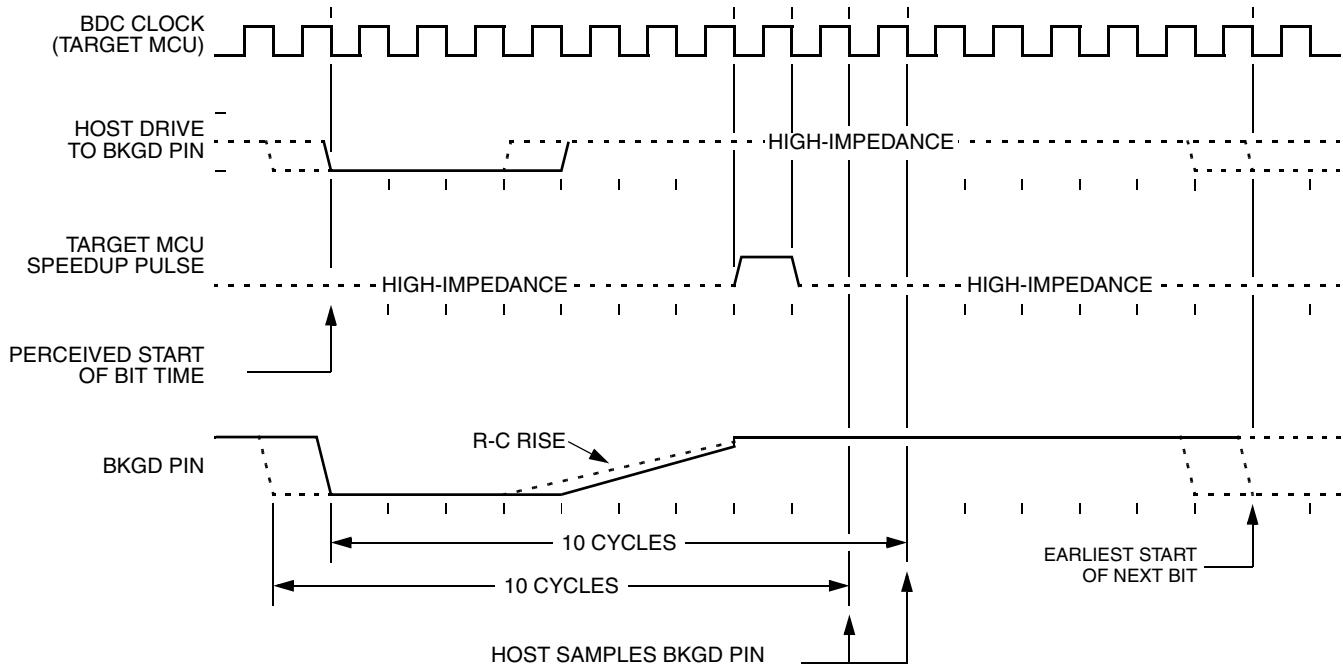


Figure 20-17. BDC Target-to-Host Serial Bit Timing (Logic 1)

Figure 20-18 shows the host receiving a logic 0 from the target MCU. Because the host is asynchronous to the target MCU, there is a 0–1 cycle delay from the host-generated falling edge on BKGD to the start of the bit time as perceived by the target MCU. The host initiates the bit time, but the target MCU finishes it. Because the target wants the host to receive a logic 0, it drives the BKGD pin low for 13 BDC clock cycles, then briefly drives it high to speed up the rising edge. The host samples the bit level about 10 cycles after starting the bit time.

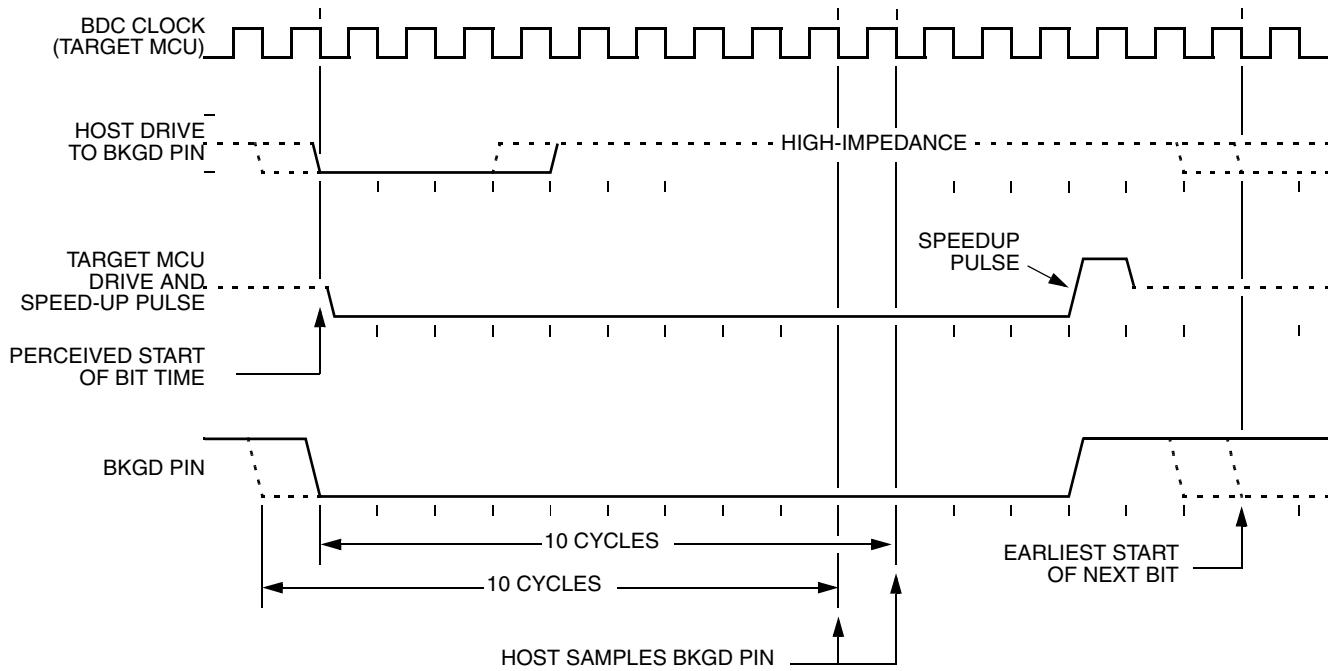


Figure 20-18. BDM Target-to-Host Serial Bit Timing (Logic 0)

#### 20.4.1.4 BDM Command Set Descriptions

This section presents detailed descriptions of the BDM commands.

The V1 BDM command set is based on transmission of one or more 8-bit data packets per operation. Each operation begins with a host-to-target transmission of an 8-bit command code packet. The command code definition broadly maps the operations into four formats as shown in [Figure 20-19](#).

**Miscellaneous Commands**

	7	6	5	4		3	2	1	0
W	0	0	R/W	0					MSCMD
R/W					Optional Command Extension Byte (Data)				

**Memory Commands**

	7	6	5	4		3	2	1	0
W	0	0	R/W	1					SZ
W if addr, R/W if data					Command Extension Bytes (Address, Data)				

**Core Register Commands**

	7	6	5	4		3	2	1	0
W		CRG	R/W						CRN
R/W					Command Extension Bytes (Data)				

**PST Trace Buffer Read Commands**

	7	6	5	4		3	2	1	0
W	0	1	0						CRN
R					Trace Buffer Data[31–24], see <a href="#">Figure 20-15</a>				
R					Trace Buffer Data[23–16], see <a href="#">Figure 20-15</a>				
R					Trace Buffer Data[15–08], see <a href="#">Figure 20-15</a>				
R					Trace Buffer Data[07–00], see <a href="#">Figure 20-15</a>				

**Figure 20-19. BDM Command Encodings**

**Table 20-24. BDM Command Field Descriptions**

<b>Field</b>	<b>Description</b>																											
5 R/W	Read/Write. 0 Command is performing a write operation. 1 Command is performing a read operation.																											
3–0 MSCMD	Miscellaneous command. Defines the miscellaneous command to be performed. 0000 No operation 0001 Display the CPU's program counter (PC) plus optional capture in the PST trace buffer 0010 Enable the BDM acknowledge communication mode 0011 Disable the BDM acknowledge communication mode 0100 Force a CPU halt (background) 1000 Resume CPU execution (go) 1101 Read/write of the debug XCSR most significant byte 1110 Read/write of the debug CSR2 most significant byte 1111 Read/write of the debug CSR3 most significant byte																											
3–2 SZ	Memory operand size. Defines the size of the memory reference. 00 8-bit byte 01 16-bit word 10 32-bit long																											
1–0 MCMD	Memory command. Defines the type of the memory reference to be performed. 00 Simple write if R/W = 0; simple read if R/W = 1 01 Write + status if R/W = 0; read + status if R/W = 1 10 Fill if R/W = 0; dump if R/W = 1 11 Fill + status if R/W = 0; dump + status if R/W = 1																											
7–6 CRG	Core register group. Defines the core register group to be referenced. 01 CPU's general-purpose registers (An, Dn) or PST trace buffer 10 Debug's control registers 11 CPU's control registers (PC, SR, VBR, CPUCR,...)																											
4–0 CRN	Core register number. Defines the specific core register (its number) to be referenced. All other CRN values are reserved. <table border="1" style="margin-left: 20px;"> <thead> <tr> <th>CRG</th> <th>CRN</th> <th>Register</th> </tr> </thead> <tbody> <tr> <td rowspan="3">01</td> <td>0x00–0x07</td> <td>D0–7</td> </tr> <tr> <td>0x08–0x0F</td> <td>A0–7</td> </tr> <tr> <td>0x10–0x1B</td> <td>PST Buffer 0–11</td> </tr> <tr> <td>10</td> <td>DRc[4:0] as described in <a href="#">Table 20-4</a></td> <td></td> </tr> <tr> <td rowspan="6">11</td> <td>0x00</td> <td>OTHER_A7</td> </tr> <tr> <td>0x01</td> <td>VBR</td> </tr> <tr> <td>0x02</td> <td>CPUCR</td> </tr> <tr> <td>0x0E</td> <td>SR</td> </tr> <tr> <td>0x0F</td> <td>PC</td> </tr> <tr> <td></td> <td></td> <td></td> </tr> </tbody> </table>	CRG	CRN	Register	01	0x00–0x07	D0–7	0x08–0x0F	A0–7	0x10–0x1B	PST Buffer 0–11	10	DRc[4:0] as described in <a href="#">Table 20-4</a>		11	0x00	OTHER_A7	0x01	VBR	0x02	CPUCR	0x0E	SR	0x0F	PC			
CRG	CRN	Register																										
01	0x00–0x07	D0–7																										
	0x08–0x0F	A0–7																										
	0x10–0x1B	PST Buffer 0–11																										
10	DRc[4:0] as described in <a href="#">Table 20-4</a>																											
11	0x00	OTHER_A7																										
	0x01	VBR																										
	0x02	CPUCR																										
	0x0E	SR																										
	0x0F	PC																										

### 20.4.1.5 BDM Command Set Summary

[Table 20-25](#) summarizes the BDM command set. Subsequent paragraphs contain detailed descriptions of each command. The nomenclature below is used in [Table 20-25](#) to describe the structure of the BDM commands.

Commands begin with an 8-bit hexadecimal command code in the host-to-target direction (most significant bit first)

/	=	separates parts of the command
d	=	delay 32 target BDC clock cycles
ad24	=	24-bit memory address in the host-to-target direction
rd8	=	8 bits of read data in the target-to-host direction
rd16	=	16 bits of read data in the target-to-host direction
rd32	=	32 bits of read data in the target-to-host direction
rd.sz	=	read data, size defined by sz, in the target-to-host direction
wd8	=	8 bits of write data in the host-to-target direction
wd16	=	16 bits of write data in the host-to-target direction
wd32	=	32 bits of write data in the host-to-target direction
wd.sz	=	write data, size defined by sz, in the host-to-target direction
ss	=	the contents of XCSR[31:24] in the target-to-host direction (STATUS)
sz	=	memory operand size (0b00 = byte, 0b01 = word, 0b10 = long)
crn	=	core register number
WS	=	command suffix signaling the operation is with status

**Table 20-25. BDM Command Summary**

Command Mnemonic	Command Classification	ACK if Enb?	Command Structure	Description
SYNC	Always Available	N/A	N/A <sup>2</sup>	Request a timed reference pulse to determine the target BDC communication speed
ACK_DISABLE	Always Available	No	0x03/d	Disable the communication handshake. This command does not issue an ACK pulse.
ACK_ENABLE	Always Available	Yes	0x02/d	Enable the communication handshake. Issues an ACK pulse after the command is executed.
BACKGROUND	Non-Intrusive	Yes	0x04/d	Halt the CPU if ENBDM is set. Otherwise, ignore as illegal command.
DUMP_MEM.sz	Non-Intrusive	Yes	(0x32+4 x sz)/d/rd.sz	Dump (read) memory based on operand size (sz). Used with READ_MEM to dump large blocks of memory. An initial READ_MEM is executed to set up the starting address of the block and to retrieve the first result. Subsequent DUMP_MEM commands retrieve sequential operands.

**Table 20-25. BDM Command Summary (continued)**

<b>Command Mnemonic</b>	<b>Command Classification</b>	<b>ACK if Enb?</b>	<b>Command Structure</b>	<b>Description</b>
DUMP_MEM.sz_WS	Non-Intrusive	No	(0x33+4 x sz)/d/ss/rd.sz	Dump (read) memory based on operand size (sz) and report status. Used with READ_MEM{_WS} to dump large blocks of memory. An initial READ_MEM{_WS} is executed to set up the starting address of the block and to retrieve the first result. Subsequent DUMP_MEM{_WS} commands retrieve sequential operands.
FILL_MEM.sz	Non-Intrusive	Yes	(0x12+4 x sz)/wd.sz/d	Fill (write) memory based on operand size (sz). Used with WRITE_MEM to fill large blocks of memory. An initial WRITE_MEM is executed to set up the starting address of the block and to write the first operand. Subsequent FILL_MEM commands write sequential operands.
FILL_MEM.sz_WS	Non-Intrusive	No	(0x13+4 x sz)/wd.sz/d/ss	Fill (write) memory based on operand size (sz) and report status. Used with WRITE_MEM{_WS} to fill large blocks of memory. An initial WRITE_MEM{_WS} is executed to set up the starting address of the block and to write the first operand. Subsequent FILL_MEM{_WS} commands write sequential operands.
GO	Non-Intrusive	Yes	0x08/d	Resume the CPU's execution <sup>3</sup>
NOP	Non-Intrusive	Yes	0x00/d	No operation
READ_CREG	Active Background	Yes	(0xE0+CRN)/d/rd32	Read one of the CPU's control registers
READ_DREG	Non-Intrusive	Yes	(0xA0+CRN)/d/rd32	Read one of the debug module's control registers
READ_MEM.sz	Non-Intrusive	Yes	(0x30+4 x sz)/ad24/d/rd.sz	Read the appropriately-sized (sz) memory value from the location specified by the 24-bit address
READ_MEM.sz_WS	Non-Intrusive	No	(0x31+4 x sz)/ad24/d/ss/rd.sz	Read the appropriately-sized (sz) memory value from the location specified by the 24-bit address and report status
READ_PSTB	Non-Intrusive	Yes	(0x40+CRN)/d/rd32	Read the requested longword location from the PST trace buffer
READ_Rn	Active Background	Yes	(0x60+CRN)/d/rd32	Read the requested general-purpose register (An, Dn) from the CPU
READ_XCSR_BYTE	Always Available	No	0x2D/rd8	Read the most significant byte of the debug module's XCSR
READ_CSR2_BYTE	Always Available	No	0x2E/rd8	Read the most significant byte of the debug module's CSR2
READ_CSR3_BYTE	Always Available	No	0x2F/rd8	Read the most significant byte of the debug module's CSR3

**Table 20-25. BDM Command Summary (continued)**

<b>Command Mnemonic</b>	<b>Command Classification</b>	<b>ACK if Enb?</b> <sup>1</sup>	<b>Command Structure</b>	<b>Description</b>
SYNC_PC	Non-Intrusive	Yes	0x01/d	Display the CPU's current PC and capture it in the PST trace buffer
WRITE_CREG	Active Background	Yes	(0xC0+CRN)/wd32/d	Write one of the CPU's control registers
WRITE_DREG	Non-Intrusive	Yes	(0x80+CRN)/wd32/d	Write one of the debug module's control registers
WRITE_MEM.sz	Non-Intrusive	Yes	(0x10+4 x sz)/ad24/wd.sz/d	Write the appropriately-sized (sz) memory value to the location specified by the 24-bit address
WRITE_MEM.sz_WS	Non-Intrusive	No	(0x11+4 x sz)/ad24/wd.sz/d/ss	Write the appropriately-sized (sz) memory value to the location specified by the 24-bit address and report status
WRITE_Rn	Active Background	Yes	(0x40+CRN)/wd32/d	Write the requested general-purpose register (An, Dn) of the CPU
WRITE_XCSR_BYTE	Always Available	No	0xD/wd8	Write the most significant byte of the debug module's XCSR
WRITE_CSR2_BYTE	Always Available	No	0xE/wd8	Write the most significant byte of the debug module's CSR2
WRITE_CSR3_BYTE	Always Available	No	0xF/wd8	Write the most significant byte of the debug module's CSR3

<sup>1</sup> This column identifies if the command generates an ACK pulse if operating with acknowledge mode enabled. See [Section 20.4.1.5.3, "ACK\\_ENABLE,"](#) for addition information.

<sup>2</sup> The SYNC command is a special operation which does not have a command code.

<sup>3</sup> If a GO command is received while the processor is not halted, it performs no operation.

### 20.4.1.5.1 SYNC

The SYNC command is unlike other BDC commands because the host does not necessarily know the correct speed to use for serial communications until after it has analyzed the response to the SYNC command.

To issue a SYNC command, the host:

1. Drives the BKGD pin low for at least 128 cycles of the slowest possible BDC clock (bus clock or device-specific alternate clock source).
2. Drives BKGD high for a brief speed-up pulse to get a fast rise time. (This speedup pulse is typically one cycle of the host clock which is as fast as the maximum target BDC clock.)
3. Removes all drive to the BKGD pin so it reverts to high impedance.
4. Listens to the BKGD pin for the sync response pulse.

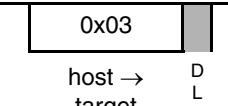
Upon detecting the sync request from the host (which is a much longer low time than would ever occur during normal BDC communications), the target:

1. Waits for BKGD to return to a logic high.

2. Delays 16 cycles to allow the host to stop driving the high speed-up pulse.
3. Drives BKGD low for 128 BDC clock cycles.
4. Drives a 1-cycle high speed-up pulse to force a fast rise time on BKGD.
5. Removes all drive to the BKGD pin so it reverts to high impedance.

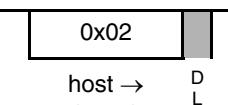
The host measures the low time of this 128-cycle sync response pulse and determines the correct speed for subsequent BDC communications. Typically, the host can determine the correct communication speed within a few percent of the actual target speed and the serial protocol can easily tolerate this speed error.

#### 20.4.1.5.2 ACK\_DISABLE

Disable host/target handshake protocol	Always Available
 <p>0x03</p> <p>host → D target L Y</p>	

Disables the serial communication handshake protocol. The subsequent commands, issued after the ACK\_DISABLE command, do not execute the hardware handshake protocol. This command is not followed by an ACK pulse.

#### 20.4.1.5.3 ACK\_ENABLE

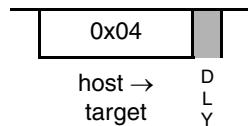
Enable host/target handshake protocol	Always Available
 <p>0x02</p> <p>host → D target L Y</p>	

Enables the hardware handshake protocol in the serial communication. The hardware handshake is implemented by an acknowledge (ACK) pulse issued by the target MCU in response to a host command. The ACK\_ENABLE command is interpreted and executed in the BDC logic without the need to interface with the CPU. However, an acknowledge (ACK) pulse is issued by the target device after this command is executed. This feature can be used by the host to evaluate if the target supports the hardware handshake protocol. If the target supports the hardware handshake protocol, subsequent commands are enabled to execute the hardware handshake protocol, otherwise this command is ignored by the target.

For additional information about the hardware handshake protocol, refer to [Section 20.4.1.6, “Serial Interface Hardware Handshake Protocol,”](#) and [Section 20.4.1.7, “Hardware Handshake Abort Procedure.”](#)

#### 20.4.1.5.4 BACKGROUND

**Enter active background mode (if enabled)** Non-intrusive



Provided XCSR[ENBDM] is set (BDM enabled), the BACKGROUND command causes the target MCU to enter active background (halt) mode as soon as the current CPU instruction finishes. If ENBDM is cleared (its default value), the BACKGROUND command is ignored.

A delay of 32 BDC clock cycles is required after the BACKGROUND command to allow the target MCU to finish its current CPU instruction and enter active background mode before a new BDC command can be accepted.

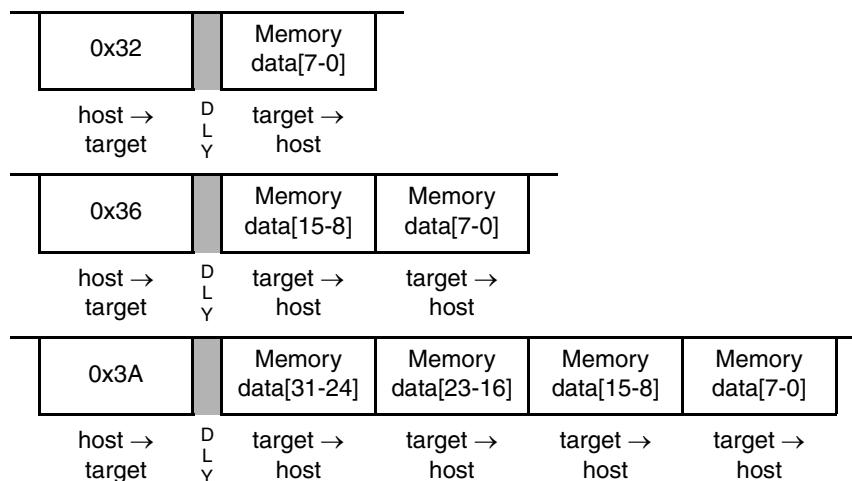
After the target MCU is reset into a normal operating mode, the host debugger would send a WRITE\_XCSR\_BYTE command to set ENBDM before attempting to send the BACKGROUND command the first time. Normally, the development host would set ENBDM once at the beginning of a debug session or after a target system reset, and then leave the ENBDM bit set during debugging operations. During debugging, the host would use GO commands to move from active background mode to normal user program execution and would use BACKGROUND commands or breakpoints to return to active background mode.

#### 20.4.1.5.5 DUMP\_MEM.sz, DUMP\_MEM.sz\_WS

**DUMP\_MEM.sz**

**Read memory specified by debug address register, then increment address**

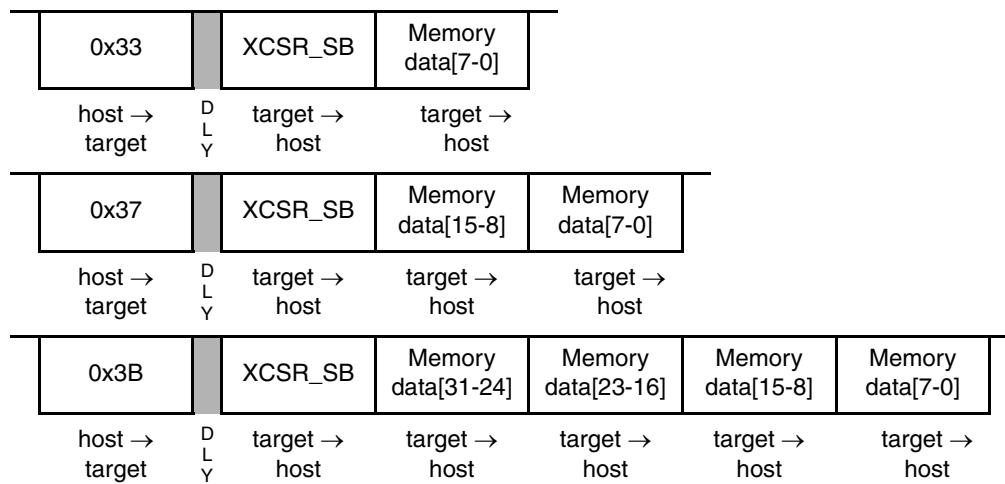
Non-intrusive



**DUMP\_MEM.sz\_WS**

**Read memory specified by debug address register with status,  
then increment address**

**Non-intrusive**



DUMP\_MEM{\_WS} is used with the READ\_MEM{\_WS} command to access large blocks of memory. An initial READ\_MEM{\_WS} is executed to set-up the starting address of the block and to retrieve the first result. If an initial READ\_MEM{\_WS} is not executed before the first DUMP\_MEM{\_WS}, an illegal command response is returned. The DUMP\_MEM{\_WS} 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\_MEM{\_WS} commands use this address, perform the memory read, increment it by the current operand size, and store the updated address in the temporary register. If the with-status option is specified, the core status byte (XCSR\_SB) contained in XCSR[31–24] is returned before the read data. XCSR\_SB reflects the state after the memory read was performed.

**NOTE**

DUMP\_MEM{\_WS} does not check for a valid address; it is a valid command only when preceded by NOP, READ\_MEM{\_WS}, or another DUMP\_MEM{\_WS} command. Otherwise, an illegal command response is returned. NOP can be used for inter-command padding without corrupting the address pointer.

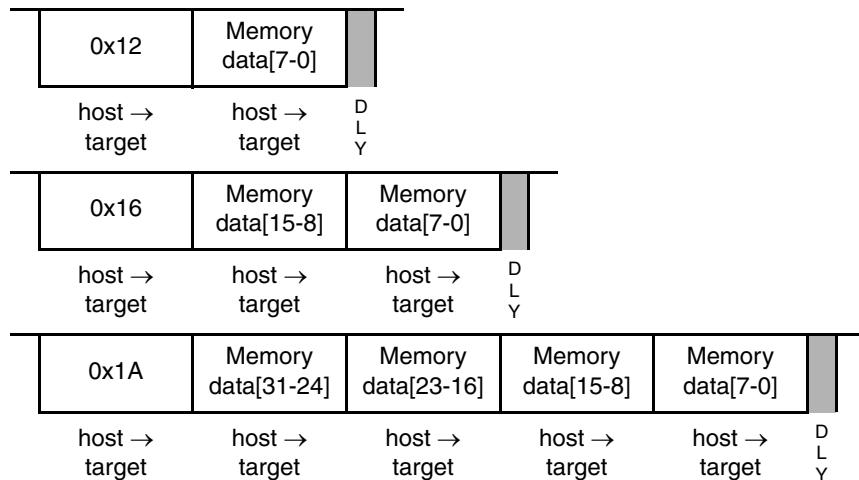
The size field (sz) is examined each time a DUMP\_MEM{\_WS} command is processed, allowing the operand size to be dynamically altered. The examples show the DUMP\_MEM.B{\_WS}, DUMP\_MEM.W{\_WS} and DUMP\_MEM.L{\_WS} commands.

### 20.4.1.5.6 FILL\_MEM.sz, FILL\_MEM.sz\_WS

#### FILL\_MEM.sz

**Write memory specified by debug address register, then increment address**

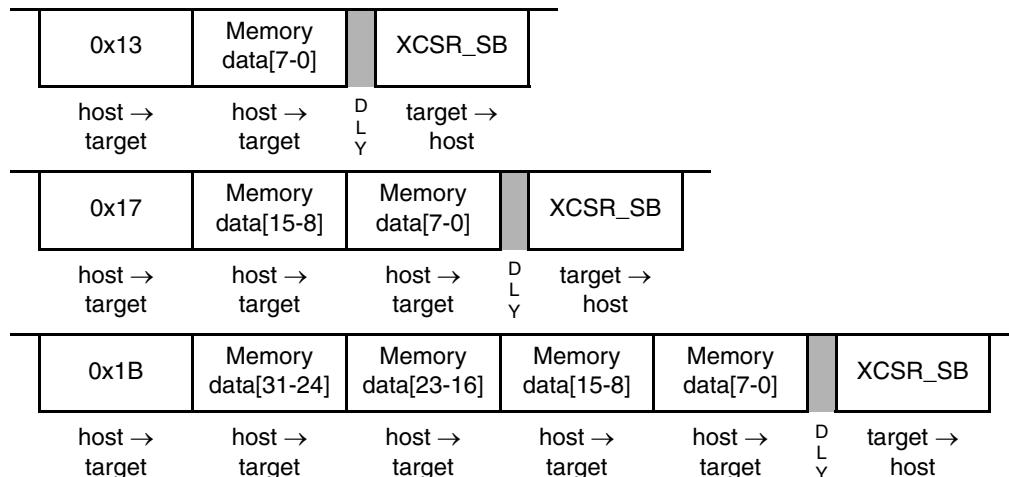
**Non-intrusive**



#### FILL\_MEM.sz\_WS

**Write memory specified by debug address register with status, then increment address**

**Non-intrusive**



FILL\_MEM{\_WS} is used with the WRITE\_MEM{\_WS} command to access large blocks of memory. An initial WRITE\_MEM{\_WS} is executed to set up the starting address of the block and write the first datum. If an initial WRITE\_MEM{\_WS} is not executed before the first FILL\_MEM{\_WS}, an illegal command response is returned. The FILL\_MEM{\_WS} command stores subsequent operands. The initial address is incremented by the operand size (1, 2, or 4) and saved in a temporary register. Subsequent WRITE\_MEM{\_WS} commands use this address, perform the memory write, increment it by the current operand size, and store the updated address in the temporary register. If the with-status option is specified,

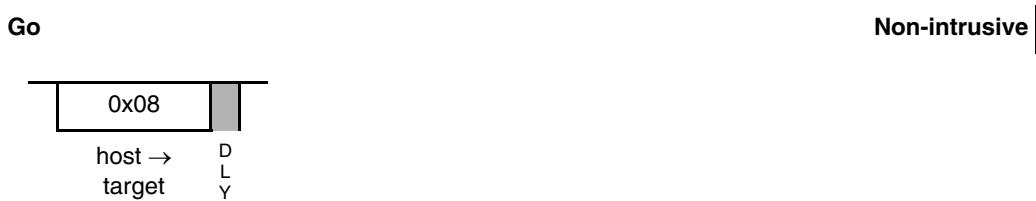
the core status byte (XCSR\_SB) contained in XCSR[31–24] is returned after the write data. XCSR\_SB reflects the state after the memory write was performed.

#### NOTE

`FILL_MEM{_WS}` does not check for a valid address; it is a valid command only when preceded by `NOP`, `WRITE_MEM{_WS}`, or another `FILL_MEM{_WS}` command. Otherwise, an illegal command response is returned. `NOP` can be used for intercommand padding without corrupting the address pointer.

The size field (sz) is examined each time a `FILL_MEM{_WS}` command is processed, allowing the operand size to be dynamically altered. The examples show the `FILL_MEM.B{_WS}`, `FILL_MEM.W{_WS}` and `FILL_MEM.L{_WS}` commands.

#### 20.4.1.5.7 GO



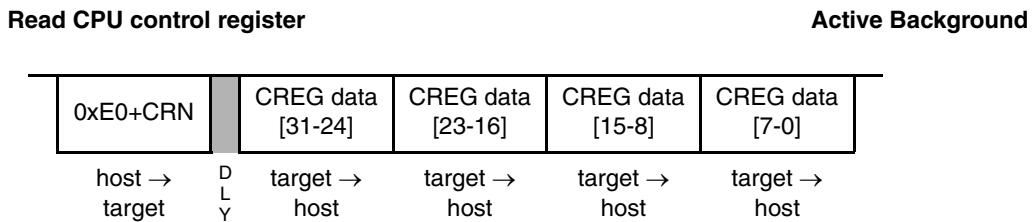
This command is used to exit active background (halt) mode and begin (or resume) execution of the application's instructions. The CPU's pipeline is flushed and refilled before normal instruction execution resumes. Prefetching begins at the current address in the PC and at the current privilege level. 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.

#### 20.4.1.5.8 NOP



NOP performs no operation and may be used as a null command where required.

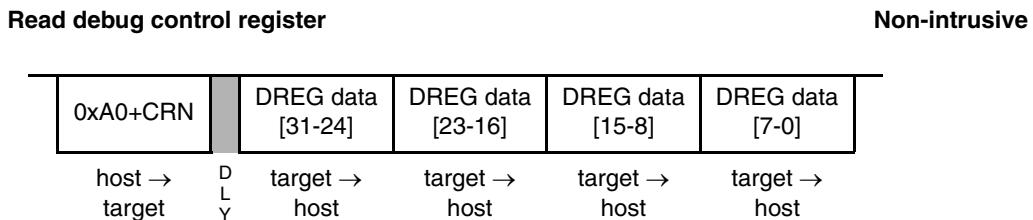
### 20.4.1.5.9 READ\_CREG



If the processor is halted, this command reads the selected control register and returns the 32-bit result. This register grouping includes the PC, SR, CPUCR, VBR, and OTHER\_A7. Accesses to processor control registers are always 32-bits wide, regardless of implemented register width. The register is addressed through the core register number (CRN). See [Table 20-24](#) for the CRN details when CRG is 11.

If the processor is not halted, this command is rejected as an illegal operation and no operation is performed.

### 20.4.1.5.10 READ\_DREG



This command reads the selected debug control register and returns the 32-bit result. This register grouping includes the CSR, XCSR, CSR2, and CSR3. Accesses to debug control registers are always 32-bits wide, regardless of implemented register width. The register is addressed through the core register number (CRN). See [Table 20-4](#) for CRN details.

### 20.4.1.5.11 READ\_MEM.sz, READ\_MEM.sz\_WS

#### READ\_MEM.sz

**Read memory at the specified address** Non-intrusive

0x30	Address[23-0]	D L Y	Memory data[7-0]			
host → target	host → target		target → host			
0x34	Address[23-0]	D L Y	Memory data[15-8]	Memory data[7-0]		
host → target	host → target		target → host	target → host		
0x38	Address[23-0]	D L Y	Memory data[31-24]	Memory data[23-16]	Memory data[15-8]	Memory data[7-0]
host → target	host → target		target → host	target → host	target → host	target → host

#### READ\_MEM.sz\_WS

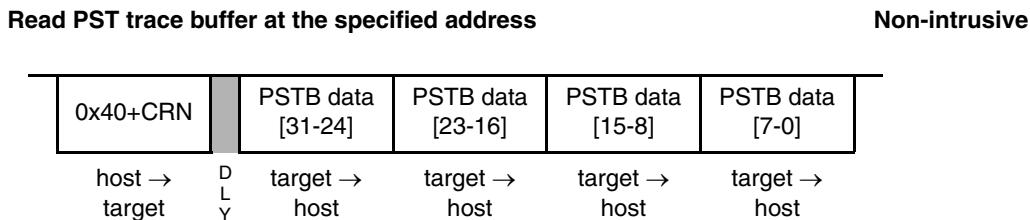
**Read memory at the specified address with status** Non-intrusive

0x31	Address[23-0]	D L Y	XCSR_SB	Memory data[7-0]			
host → target	host → target		target → host	target → host			
0x35	Address[23-0]	D L Y	XCSR_SB	Memory data [15-8]	Memory data [7-0]		
host → target	host → target		target → host	target → host	target → host		
0x39	Address[23-0]	D L Y	XCSR_SB	Memory data[31-24]	Memory data[23-16]	Memory data [15-8]	Memory data [7-0]
host → target	host → target		target → host	target → host	target → host	target → host	target → host

Read data at the specified memory address. The reference address is transmitted as three 8-bit packets (msb to lsb) immediately after the command packet. The access attributes are defined by BAAR[TT,TM]. The hardware forces low-order address bits to zeros for word and longword accesses to ensure these accesses are on 0-modulo-size alignments. If the with-status option is specified, the core status byte (XCSR\_SB) contained in XCSR[31–24] is returned before the read data. XCSR\_SB reflects the state after the memory read was performed.

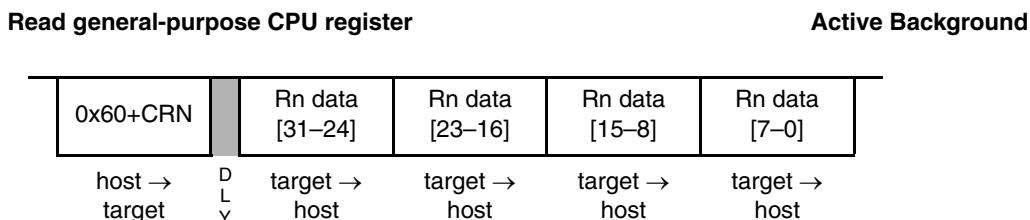
The examples show the READ\_MEM.B{\_WS}, READ\_MEM.W{\_WS} and READ\_MEM.L{\_WS} commands.

#### 20.4.1.5.12 READ\_PSTB



Read 32 bits of captured PST/DDATA values from the trace buffer at the specified address. The PST trace buffer contains 64 six-bit entries, packed consecutively into 12 longword locations. See [Figure 20-15](#) for an illustration of how the buffer entries are packed.

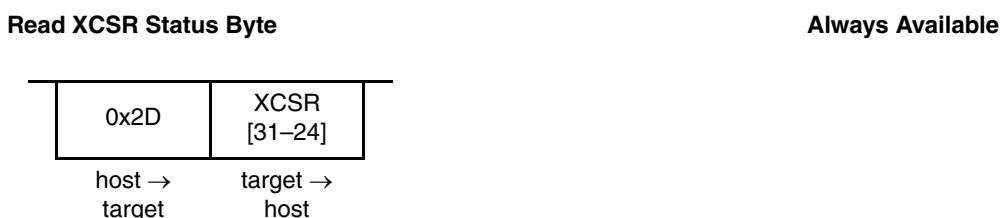
#### 20.4.1.5.13 READ\_Rn



If the processor is halted, this command reads the selected CPU general-purpose register (An, Dn) and returns the 32-bit result. See [Table 20-24](#) for the CRN details when CRG is 01.

If the processor is not halted, this command is rejected as an illegal operation and no operation is performed.

#### 20.4.1.5.14 READ\_XCSR\_BYTE



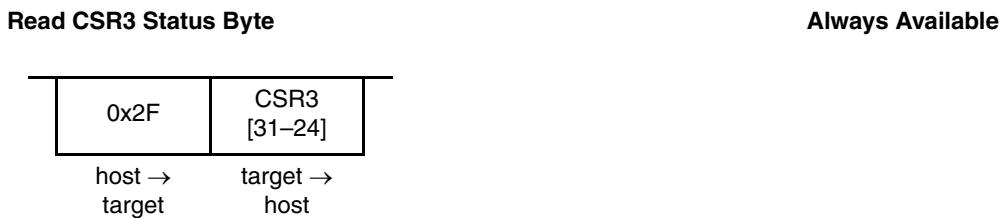
Read the special status byte of XCSR (XCSR[31-24]). This command can be executed in any mode.

### 20.4.1.5.15 READ\_CSR2\_BYTE



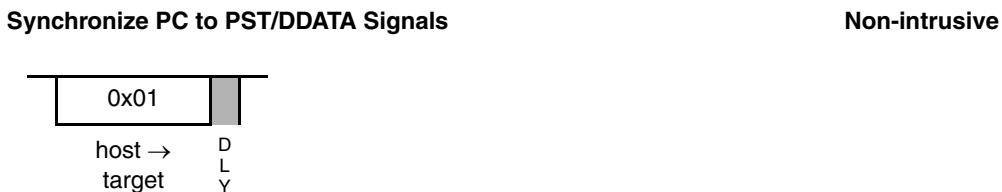
Read the most significant byte of CSR2 (CSR2[31–24]). This command can be executed in any mode.

### 20.4.1.5.16 READ\_CSR3\_BYTE



Read the most significant byte of the CSR3 (CSR3[31–24]). This command can be executed in any mode.

### 20.4.1.5.17 SYNC\_PC



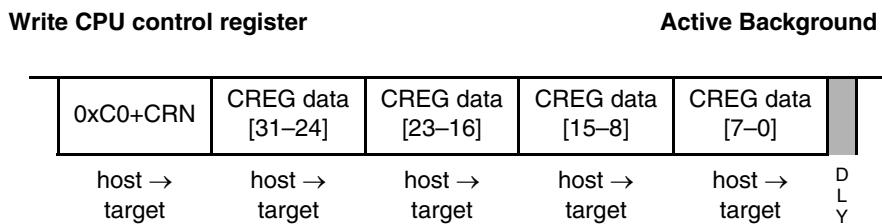
Capture the processor's current PC (program counter) and display it on the PST/DDATA signals. After the debug module receives the command, it sends a signal to the ColdFire core that the current PC must be displayed. The core responds by forcing an instruction fetch to the next PC with the address being captured by the DDATA logic. The DDATA logic captures a 2- or 3-byte instruction address, based on CSR[9]. If CSR[9] is cleared, then a 2-byte address is captured, else a 3-byte address is captured. The specific sequence of PST and DDATA values is defined as:

1. Debug signals a SYNC\_PC command is pending.
2. CPU completes the current instruction.
3. CPU forces an instruction fetch to the next PC, generating a PST = 0x5 value indicating a taken branch. DDATA captures the instruction address corresponding to the PC. DDATA generates a PST marker signalling a 2- or 3-byte address as defined by CSR[9] (CSR[9] = 0, 2-byte; CSR[9] = 1, 3-byte) and displays the captured PC address.

This command can be used to provide a PC synchronization point between the core's execution and the application code in the PST trace buffer. It can also be used to dynamically access the PC for performance

monitoring as the execution of this command is considerably less obtrusive to the real-time operation of an application than a BACKGROUND/read-PC/GO command sequence.

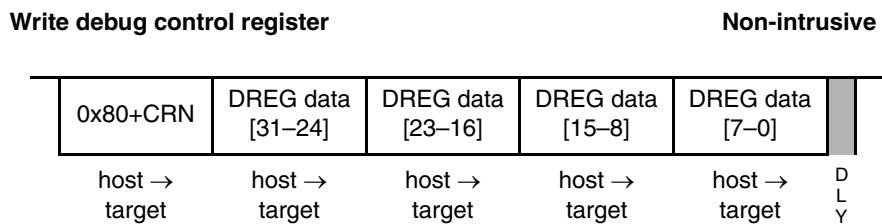
#### 20.4.1.5.18 WRITE\_CREG



If the processor is halted, this command writes the 32-bit operand to the selected control register. This register grouping includes the PC, SR, CPUCR, VBR, and OTHER\_A7. Accesses to processor control registers are always 32-bits wide, regardless of implemented register width. The register is addressed through the core register number (CRN). See [Table 20-24](#) for the CRN details when CRG is 11.

If the processor is not halted, this command is rejected as an illegal operation and no operation is performed.

#### 20.4.1.5.19 WRITE\_DREG



This command writes the 32-bit operand to the selected debug control register. This grouping includes all the debug control registers ({X}CSR $n$ , BAAR, AATR, TDR, PBR $n$ , PBMR, ABxR, DBR, DBMR). Accesses to debug control registers are always 32-bits wide, regardless of implemented register width. The register is addressed through the core register number (CRN). See [Table 20-4](#) for CRN details.

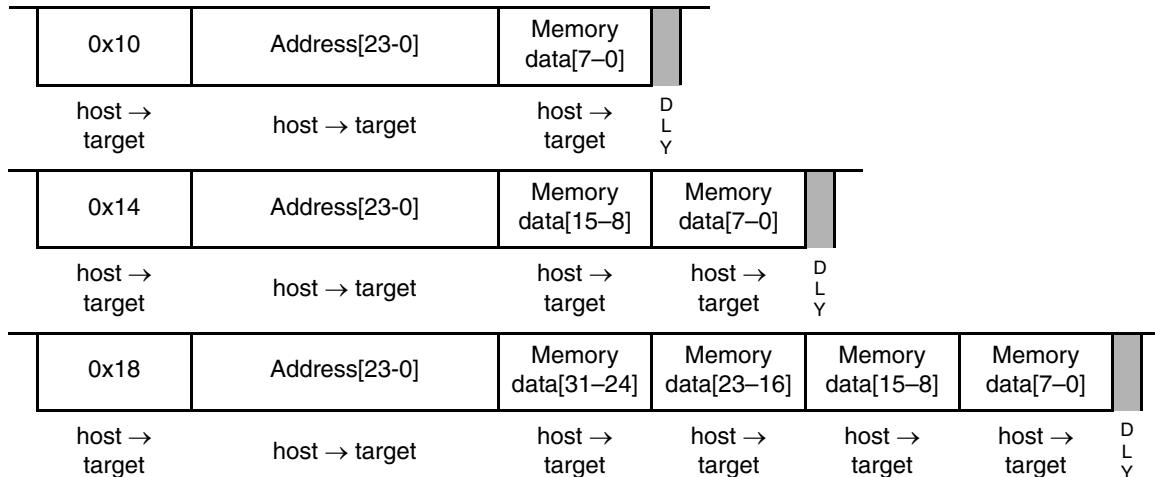
#### NOTE

When writing XCSR, CSR2, or CSR3, WRITE\_DREG only writes bits 23–0. The upper byte of these debug registers is only written with the special WRITE\_XCSR\_BYTEx, WRITE\_CSR2\_BYTEx, and WRITE\_CSR3\_BYTEx commands.

### 20.4.1.5.20 WRITE\_MEM.sz, WRITE\_MEM.sz\_WS

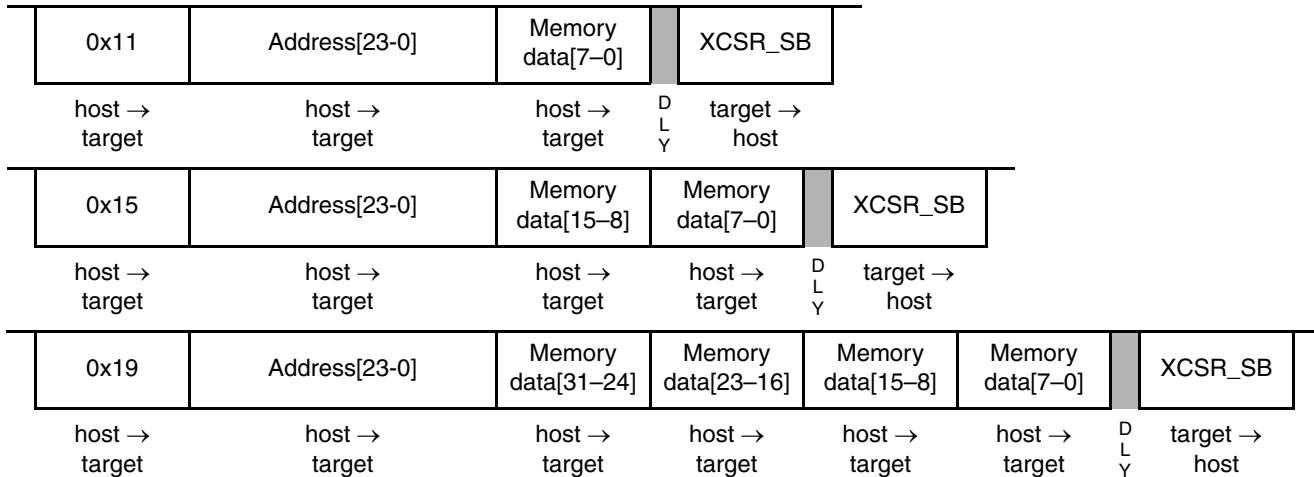
#### WRITE\_MEM.sz

**Write memory at the specified address** Non-intrusive



#### WRITE\_MEM.sz\_WS

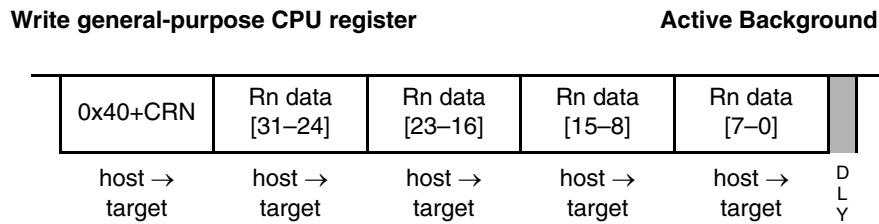
**Write memory at the specified address with status** Non-intrusive



Write data at the specified memory address. The reference address is transmitted as three 8-bit packets (msb to lsb) immediately after the command packet. The access attributes are defined by BAAR[TT,TM]. The hardware forces low-order address bits to zeros for word and longword accesses to ensure these accesses are on 0-modulo-size alignments. If the with-status option is specified, the core status byte (XCSR\_SB) contained in XCSR[31–24] is returned after the read data. XCSR\_SB reflects the state after the memory write was performed.

The examples show the WRITE\_MEM.B{\_WS}, WRITE\_MEM.W{\_WS}, and WRITE\_MEM.L{\_WS} commands.

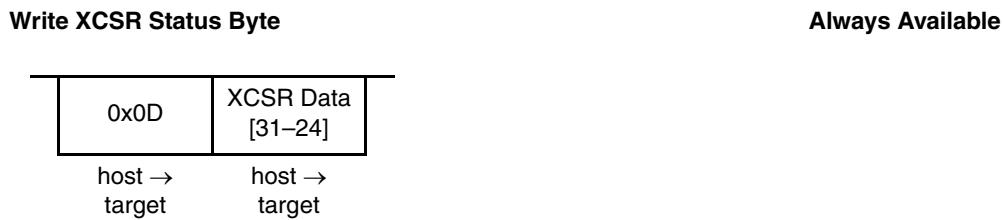
#### 20.4.1.5.21 WRITE\_Rn



If the processor is halted, this command writes the 32-bit operand to the selected CPU general-purpose register (An, Dn). See [Table 20-24](#) for the CRN details when CRG is 01.

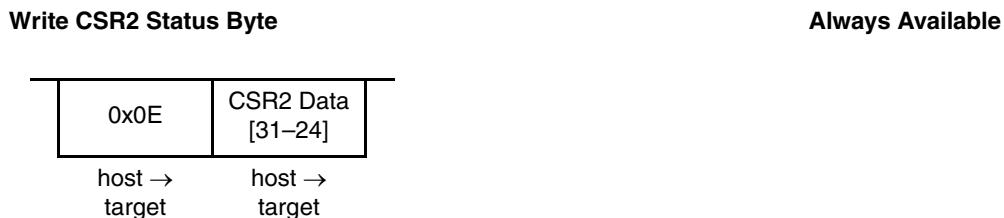
If the processor is not halted, this command is rejected as an illegal operation and no operation is performed.

#### 20.4.1.5.22 WRITE\_XCSR\_BYTE



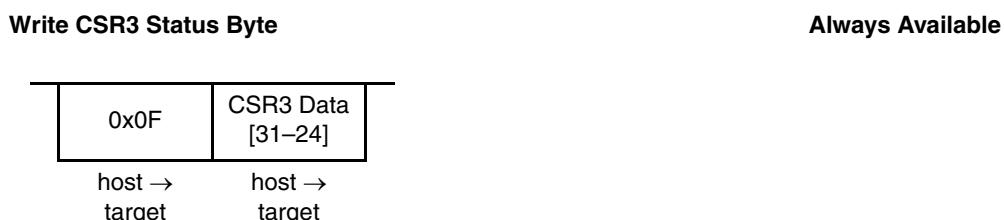
Write the special status byte of XCSR (XCSR[31–24]). This command can be executed in any mode.

#### 20.4.1.5.23 WRITE\_CSR2\_BYTE



Write the most significant byte of CSR2 (CSR2[31–24]). This command can be executed in any mode.

#### 20.4.1.5.24 WRITE\_CSR3\_BYTE

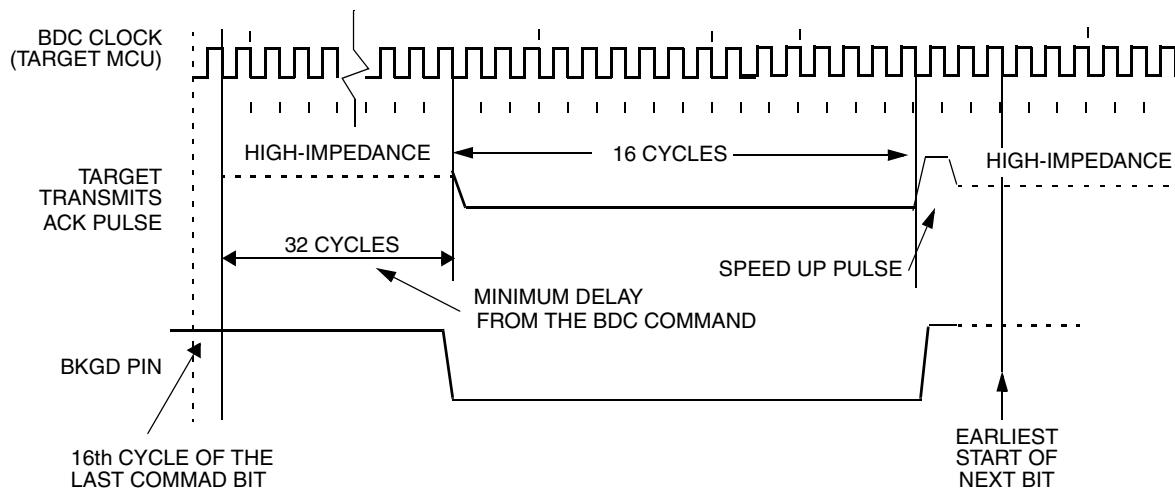


Write the most significant byte of CSR3 (CSR3[31–24]). This command can be executed in any mode.

### 20.4.1.6 Serial Interface Hardware Handshake Protocol

BDC commands that require CPU execution are ultimately treated at the core clock rate. Because the BDC clock source can be asynchronous relative to the bus frequency when CLKSW is cleared, it is necessary to provide a handshake protocol so the host can determine when an issued command is executed by the CPU. This section describes this protocol.

The hardware handshake protocol signals to the host controller when an issued command was successfully executed by the target. This protocol is implemented by a low pulse (16 BDC clock cycles) followed by a brief speedup pulse on the BKGD pin, generated by the target MCU when a command, issued by the host, has been successfully executed. See [Figure 20-20](#). This pulse is referred to as the ACK pulse. After the ACK pulse is finished, the host can start the data-read portion of the command if the last-issued command was a read command, or start a new command if the last command was a write command or a control command (BACKGROUND, GO, NOP, SYNC\_PC). The ACK pulse is not issued earlier than 32 BDC clock cycles after the BDC command was issued. The end of the BDC command is assumed to be the 16th BDC clock cycle of the last bit. This minimum delay assures enough time for the host to recognize the ACK pulse. There is no upper limit for the delay between the command and the related ACK pulse, because the command execution depends on the CPU bus frequency, which in some cases could be slow compared to the serial communication rate. This protocol allows great flexibility for pod designers, because it does not rely on any accurate time measurement or short response time to any event in the serial communication.



**Figure 20-20. Target Acknowledge Pulse (ACK)**

#### NOTE

If the ACK pulse was issued by the target, the host assumes the previous command was executed. If the CPU enters a stop mode prior to executing a non-intrusive command, the command is discarded and the ACK pulse is not issued. After entering a stop mode, the BDC command is no longer pending and the XCSR[CSTAT] value of 001 is kept until the next command is successfully executed.

Figure 20-21 shows the ACK handshake protocol in a command level timing diagram. A READ\_MEM.B command is used as an example:

1. The 8-bit command code is sent by the host, followed by the address of the memory location to be read.
2. The target BDC decodes the command and sends it to the CPU.
3. Upon receiving the BDC command request, the CPU schedules a execution slot for the command.
4. The CPU temporarily stalls the instruction stream at the scheduled point, executes the READ\_MEM.B command and then continues.

This process is referred to as cycle stealing. The READ\_MEM.B appears as a single-cycle operation to the processor, even though the pipelined nature of the Operand Execution Pipeline requires multiple CPU clock cycles for it to actually complete. After that, the debug module tracks the execution of the READ\_MEM.B command as the processor resumes the normal flow of the application program. After detecting the READ\_MEM.B command is done, the BDC issues an ACK pulse to the host controller, indicating that the addressed byte is ready to be retrieved. After detecting the ACK pulse, the host initiates the data-read portion of the command.

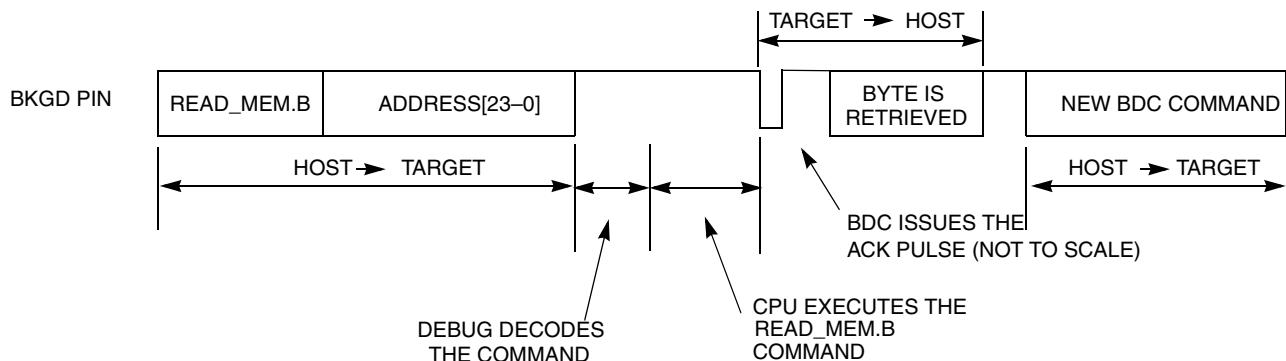


Figure 20-21. Handshake Protocol at Command Level

Unlike a normal bit transfer, where the host initiates the transmission by issuing a negative edge in the BKGD pin, the serial interface ACK handshake pulse is initiated by the target MCU. The hardware handshake protocol in Figure 20-21 specifies the timing when the BKGD pin is being driven, so the host should follow these timing relationships to avoid the risks of an electrical conflict at the BKGD pin.

The ACK handshake protocol does not support nested ACK pulses. If a BDC command is not acknowledged by an ACK pulse, the host first needs to abort the pending command before issuing a new BDC command. When the CPU enters a stop mode at about the same time the host issues a command that requires CPU execution, the target discards the incoming command. Therefore, the command is not acknowledged by the target, meaning that the ACK pulse is not issued in this case. After a certain time, the host could decide to abort the ACK protocol to allow a new command. Therefore, the protocol provides a mechanism where a command (a pending ACK) could be aborted. Unlike a regular BDC command, the ACK pulse does not provide a timeout. In the case of a STOP instruction where the ACK is prevented from being issued, it would remain pending indefinitely if not aborted. See the handshake abort procedure described in Section 20.4.1.7, “Hardware Handshake Abort Procedure.”

### 20.4.1.7 Hardware Handshake Abort Procedure

The abort procedure is based on the SYNC command. To abort a command that has not responded with an ACK pulse, the host controller generates a sync request (by driving BKGD low for at least 128 serial clock cycles and then driving it high for one serial clock cycle as a speedup pulse). By detecting this long low pulse on the BKGD pin, the target executes the sync protocol (see [Section 20.4.1.5.1, “SYNC”](#)), and assumes that the pending command and therefore the related ACK pulse, are being aborted. Therefore, after the sync protocol completes, the host is free to issue new BDC commands.

Because the host knows the target BDC clock frequency, the SYNC command does not need to consider the lowest possible target frequency. In this case, the host could issue a SYNC close to the 128 serial clock cycles length, providing a small overhead on the pulse length to assure the sync pulse is not misinterpreted by the target.

It is important to notice that any issued BDC command that requires CPU execution is scheduled for execution by the pipeline based on the dynamic state of the machine, provided the processor does not enter any of the stop modes. If the host aborts a command by sending the sync pulse, it should then read XCSR[CSTAT] after the sync response is issued by the target, checking for CSTAT cleared, before attempting to send any new command that requires CPU execution. This prevents the new command from being discarded at the debug/CPU interface, due to the pending command being executed by the CPU. Any new command should be issued only after XCSR[CSTAT] is cleared.

There are multiple reasons that could cause a command to take too long to execute, measured in terms of the serial communication rate: The BDC clock frequency could be much faster than the CPU clock frequency, or the CPU could be accessing a slow memory, which would cause pipeline stall cycles to occur. All commands referencing the CPU registers or memory require access to the processor’s local bus to complete. If the processor is executing a tight loop contained within a single aligned longword, the processor may never successfully grant the internal bus to the debug command. For example:

```
align    4
label1: nop
bra.b   label1
or

align    4
label2: bra.w  label2
```

These two examples of tight loops exhibit the BDM lockout behavior. If the loop spans across two longwords, there are no issues, so the recommended construct is:

```
align    4
label3: bra.l  label3
```

The hardware handshake protocol is appropriate for these situations, but the host could also decide to use the software handshake protocol instead. In this case, if XCSR[CSTAT] is 001, there is a BDC command pending at the debug/CPU interface. The host controller should monitor XCSR[CSTAT] and wait until it is 000 to be able to issue a new command that requires CPU execution. However, if the XCSR[CSTAT] is 1xx, the host should assume the last command failed to execute. To recover from this condition, the following sequence is suggested:

1. Issue a SYNC command to reset the BDC communication channel.
2. The host issues a BDM NOP command.

3. The host reads the channel status using a READ\_XCSR\_BYT command.
4. If XCSR[CSTAT] is 000  
then the status is okay; proceed  
else

Halt the CPU using a BDM BACKGROUND command

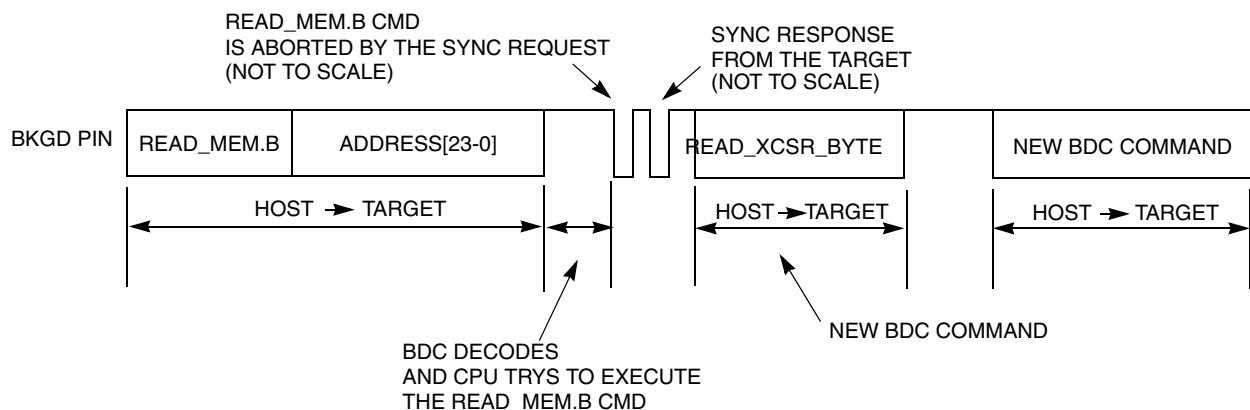
Repeat steps 1,2,3

If XCSR[CSTAT] is 000, then proceed, else reset the device

**Figure 20-22** shows a SYNC command aborting a READ\_MEM.B. After the command is aborted, a new command could be issued by the host.

### NOTE

**Figure 20-22** signal timing is not drawn to scale.



**Figure 20-22. ACK Abort Procedure at the Command Level**

**Figure 20-23** a shows a conflict between the ACK pulse and the sync request pulse. This conflict could occur if a pod device is connected to the target BKGD pin and the target is already executing a BDC command. Consider that the target CPU is executing a pending BDC command at the exact moment the pod is being connected to the BKGD pin. In this case, an ACK pulse is issued at the same time as the SYNC command. In this case there is an electrical conflict between the ACK speedup pulse and the sync pulse. Because this is not a probable situation, the protocol does not prevent this conflict from happening.

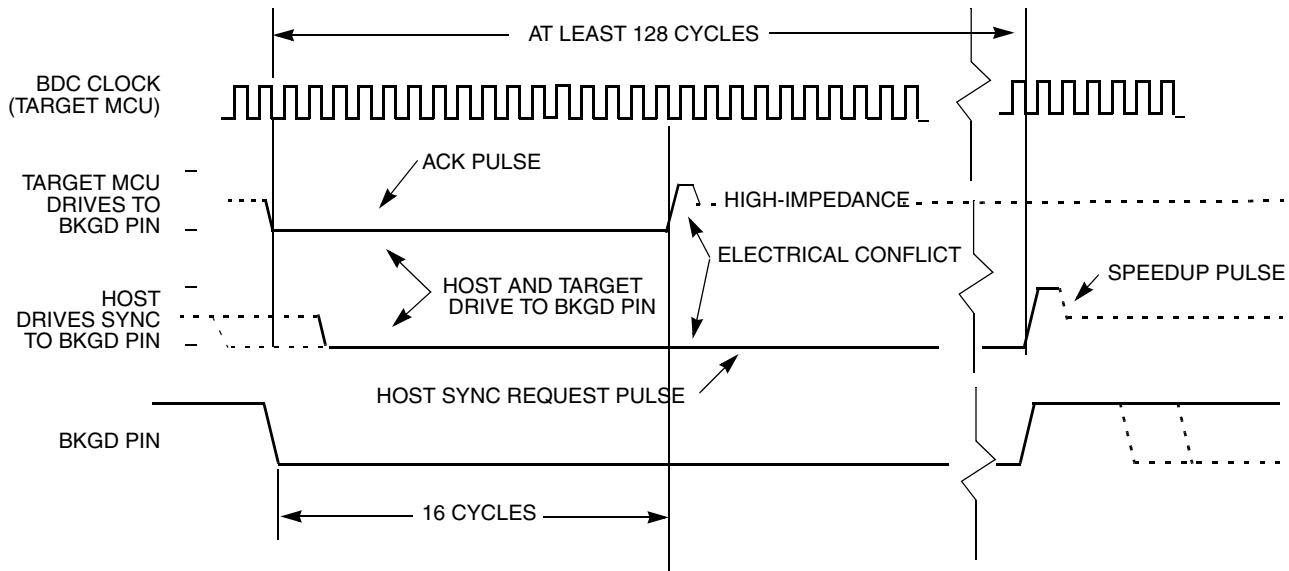


Figure 20-23. ACK Pulse and SYNC Request Conflict

The hardware handshake protocol is enabled by the `ACK_ENABLE` command and disabled by the `ACK_DISABLE` command. It also allows for pod devices to choose between the hardware handshake protocol or the software protocol that monitors the XCSR status byte. The `ACK_ENABLE` and `ACK_DISABLE` commands are:

- `ACK_ENABLE` — Enables the hardware handshake protocol. The target issues the ACK pulse when a CPU command is executed. The `ACK_ENABLE` command itself also has the ACK pulse as a response.
- `ACK_DISABLE` — Disables the ACK pulse protocol. In this case, the host should verify the state of XCSR[CSTAT] to evaluate if there are pending commands and to check if the CPU's operating state has changed to or from active background mode via XCSR[31–30].

The default state of the protocol, after reset, is hardware handshake protocol disabled.

The commands that do not require CPU execution, or that have the status register included in the retrieved bit stream, do not perform the hardware handshake protocol. Therefore, the target does not respond with an ACK pulse for those commands even if the hardware protocol is enabled. Conversely, only commands that require CPU execution and do not include the status byte perform the hardware handshake protocol. See the third column in [Table 20-25](#) for the complete enumeration of this function.

An exception is the `ACK_ENABLE` command, which does not require CPU execution but responds with the ACK pulse. This feature can be used by the host to evaluate if the target supports the hardware handshake protocol. If an ACK pulse is issued in response to this command, the host knows that the target supports the hardware handshake protocol. If the target does not support the hardware handshake protocol the ACK pulse is not issued. In this case, the `ACK_ENABLE` command is ignored by the target, because it is not recognized as a valid command.

## 20.4.2 Real-Time Debug Support

The ColdFire family supports 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 the small intrusions with minimal effect on real-time operation.

### NOTE

The details regarding real-time debug support will be supplied at a later time.

## 20.4.3 Trace Support

For the baseline V1 ColdFire core and its single debug signal, support for trace functionality is completely redefined. The V1 solution provides an on-chip PST/DDATA trace buffer (known as the PSTB) to record the stream of PST and DDATA values.

As a review, the classic ColdFire debug architecture supports real-time trace via the PST/DDATA output signals. For this functionality, the following apply:

- One (or more) PST value is generated for each executed instruction
- Branch target instruction address information is displayed on all non-PC-relative change-of-flow instructions, where the user selects a programmable number of bytes of target address
  - Displayed information includes PST marker plus target instruction address as DDATA
  - Captured address creates the appropriate number of DDATA entries, each with 4 bits of address
- Optional data trace capabilities are provided for accesses mapped to the slave peripheral bus
  - Displayed information includes PST marker plus captured operand value as DDATA
  - Captured operand creates the appropriate number of DDATA entries, each with 4 bits of data

The resulting PST/DDATA output stream, with the application program memory image, provides an instruction-by-instruction dynamic trace of the execution path.

Even with the application of a PST trace buffer, problems associated with the PST bandwidth and associated fill rate of the buffer remain. Given that there is one (or more) PST entry per instruction, the PSTB would fill rapidly without some type of data compression.

Consider the following example to illustrate the PST compression algorithm. Most sequential instructions generate a single  $PST = 1$  value. Without compression, the execution of ten sequential instructions generates a stream of ten  $PST = 1$  values. With PST compression, the reporting of any  $PST = 1$  value is delayed so that consecutive  $PST = 1$  values can be accumulated. When a  $PST \neq 1$  value is reported, the maximum accumulation count is reached, or a debug data value is captured, a single accumulated PST value is generated. Returning to the example with compression enabled, the execution of ten sequential instructions generates a single PST value indicating ten sequential instructions have been executed.

This technique has proven to be effective at significantly reducing the average PST entries per instruction and PST entries per machine cycle. The application of this compression technique makes the application of a useful PST trace buffer for the V1 ColdFire core realizable. The resulting 5-bit PST definitions are shown in [Table 20-26](#).

**Table 20-26. CF1 Debug Processor Status Encodings**

PST[4:0]	Definition
0x00	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 with this encoding.
0x01	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.
0x02	Reserved
0x03	Entry into user-mode. Signaled after execution of the instruction that caused the ColdFire processor to enter user mode.
0x04	Begin execution of PULSE and WDDATA instructions. PULSE defines triggers or markers 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 0x04 is signaled on the PST port, followed by the appropriate marker, and then the data transfer on the DDATA port. The number of captured data bytes depends on the WDDATA operand size.
0x05	Begin execution of taken branch or SYNC_PC BDM command. 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. This encoding also indicates that the SYNC_PC command has been processed.
0x06	Reserved
0x07	Begin execution of return from exception (RTE) instruction.
0x08–0x0B	Indicates the number of data bytes to be loaded into the PST trace buffer. The capturing of peripheral bus data references is controlled by CSR[DDC]. 0x08 Begin 1-byte data transfer on DDATA 0x09 Begin 2-byte data transfer on DDATA 0x0A Reserved 0x0B Begin 4-byte data transfer on DDATA
0x0C–0x0F	Indicates the number of address bytes to be loaded into the PST trace buffer. The capturing of branch target addresses is controlled by CSR[BTB]. 0x0C Reserved 0x0D Begin 2-byte address transfer on DDATA (Displayed address is shifted right 1: ADDR[16:1]) 0x0E Begin 3-byte address transfer on DDATA (Displayed address is shifted right 1: ADDR[23:1]) 0x0F Reserved
0x10–0x11	Reserved
0x12	Completed execution of 2 sequential instructions
0x13	Completed execution of 3 sequential instructions
0x14	Completed execution of 4 sequential instructions
0x15	Completed execution of 5 sequential instructions
0x16	Completed execution of 6 sequential instructions
0x17	Completed execution of 7 sequential instructions
0x18	Completed execution of 8 sequential instructions
0x19	Completed execution of 9 sequential instructions
0x1A	Completed execution of 10 sequential instructions

**Table 20-26. CF1 Debug Processor Status Encodings (continued)**

PST[4:0]	Definition
0x1B	This value signals there has been a change in the breakpoint trigger state machine. It appears as a single marker for each state change and is immediately followed by a DDATA value signaling the new breakpoint trigger state encoding. The DDATA breakpoint trigger state value is defined as $(0x20 + 2 \times \text{CSR[BSTAT]})$ : 0x20 No breakpoints enabled 0x22 Waiting for a level-1 breakpoint 0x24 Level-1 breakpoint triggered 0x2A Waiting for a level-2 breakpoint 0x2C Level-2 breakpoint triggered
0x1C	Exception processing. This value signals the processor has encountered an exception condition. Although this is a multi-cycle mode, there are only two PST = 0x1C values recorded before the mode value is suppressed.
0x1D	Emulator mode exception processing. This value signals the processor has encountered a debug interrupt or a properly-configured trace exception. Although this is a multi-cycle mode, there are only two PST = 0x1D values recorded before the mode value is suppressed.
0x1E	Processor is stopped. This value signals the processor has executed a STOP instruction. Although this is a multi-cycle mode because the ColdFire processor remains stopped until an interrupt or reset occurs, there are only two PST = 0x1E values recorded before the mode value is suppressed.
0x1F	Processor is halted. This value signals the processor has been halted. Although this is a multi-cycle mode because the ColdFire processor remains halted until a BDM go command is received or reset occurs, there are only two PST = 0x1F values recorded before the mode value is suppressed.

#### 20.4.3.1 Begin Execution of Taken Branch (PST = 0x05)

The PST is 0x05 when a taken branch is executed. For some opcodes, a branch target address may be loaded into the trace buffer (PSTB) depending on the CSR settings. CSR also controls the number of address bytes loaded that is indicated by the PST marker value immediately preceding the DDATA entry in the PSTB that begins the address entries.

Multiple byte DDATA values are displayed in least-to-most-significant order. The processor captures only those target addresses associated with taken branches that use a variant addressing mode (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 ColdFire processor loads the PSTB as follows:

1. Load PST=0x05 to identify that a taken branch is executed.
2. Optionally load the marker for the target address capture. Encodings 0x0D or 0x0E identify the number of bytes loaded into the PSTB.
3. The new target address is optionally available in the PSTB. The number of bytes of the target address loaded is configurable (2 or 3 bytes, where the encoding is 0x0D and 0x0E, respectively).

Another example of a variant branch instruction is a JMP (A0) instruction. [Figure 20-24](#) shows the PSTB entries that indicate a JMP (A0) execution, assuming CSR[BTB] was programmed to display the lower two bytes of an address.

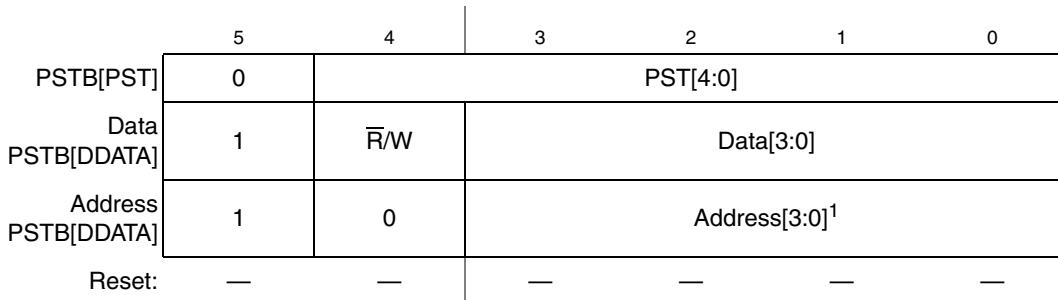
PST/DDATA Values	Description
0x05	Taken Branch
0x0D	2-byte Address Marker
{10, Address[4:1]}	Address >> 1
{10, Address[8:5]}	
{10, Address[12:9]}	
{10, Address[16:13]}	

**Figure 20-24. Example JMP Instruction Output in PSTB**

The PST of 0x05 indicates a taken branch and the marker value 0x0D indicates a 2-byte address. Therefore, the following entries display the lower two bytes of address register A0, right-shifted by 1, in least-to-most-significant nibble order. The next PST entry after the JMP instruction completes depends on the target instruction. See [Section 20.4.3.2, “PST Trace Buffer \(PSTB\) Entry Format,”](#) for entry descriptions explaining the 2-bit prefix before each address nibble.

### 20.4.3.2 PST Trace Buffer (PSTB) Entry Format

As PST and DDATA values are captured and loaded in the trace buffer, each entry is six bits in size therefore, the type of the entry can easily be determined when post-processing the PSTB. See [Figure 20-25](#).



<sup>1</sup> Depending on which nibble is displayed (as determined by CSR[9:8]), Address[3:0] sequentially (least-to-most-significant nibble order) displays four bits of the real CPU address [16:1] or [24:1].

**Figure 20-25. V1 PST/DDATA Trace Buffer Entry Format**

### 20.4.3.3 PST/DDATA Example

In this section, an example showing the behavior of the PST/DDATA functionality is detailed. Consider the following interrupt service routine that counts the interrupt, negates the IRQ, performs a software IACK, and then exits. This example is presented here because it exercises a considerable set of the PST/DDATA capabilities.

```
_isr:
```

## Version 1 ColdFire Debug (CF1\_DEBUG)

```

01074: 46fc 2700      mov.w  &0x2700,%sr      # disable interrupts
01078: 2f08          mov.l  %a0,-(%sp)    # save a0
0107a: 2f00          mov.l  %d0,-(%sp)    # save d0
0107c: 302f 0008      mov.w  (8,%sp),%d0    # load format/vector word
01080: e488          lsr.l  &2,%d0       # align vector number
01082: 0280 0000 00ff  andi.l &0xff,%d0     # isolate vector number
01088: 207c 0080 1400  mov.l  &int_count,%a0   # base of interrupt counters

_isr_entry1:
0108e: 52b0 0c00      addq.l &1,(0,%a0,%d0.1*4) # count the interrupt
01092: 11c0 a021      mov.b  %d0,IGCR0+1.w  # negate the irq
01096: 1038 a020      mov.b  IGCRO.w,%d0    # force the write to complete
0109a: 4e71          nop                # synchronize the pipelines
0109c: 71b8 ffe0      mvz.b  SWIACK.w,%d0    # software iack: pending irq?
010a0: 0c80 0000 0041  cmpi.l %d0,&0x41     # level 7 or none pending?
010a6: 6f08          ble.b  _isr_exit     # yes, then exit
010a8: 52b9 0080 145c  addq.l &1,swiack_count # increment the swiack count
010ae: 60de          bra.b  _isr_entry1   # continue at entry1

_isr_exit:
010b0: 201f          mov.l  (%sp)+,%d0    # restore d0
010b2: 205f          mov.l  (%sp)+,%a0    # restore a0
010b4: 4e73          rte                # exit

```

This ISR executes mostly as straight-line code: there is a single conditional branch @ PC = 0x10A6, which is taken in this example. The following description includes the PST and DDATA values generated as this code snippet executes. In this example, the CSR setting enables only the display of 2-byte branch addresses. Operand data captures are not enabled. The sequence begins with an interrupt exception:

```

interrupt exception occurs @ pc = 5432 while in user mode
                                                # pst      = 1c, 1c, 05, 0d
                                                # ddata   = 2a, 23, 28, 20
                                                #           trg_addr = 083a << 1
                                                #           trg_addr = 1074

_isr:
01074: 46fc 2700      mov.w  &0x2700,%sr      # pst      = 01
01078: 2f08          mov.l  %a0,-(%sp)    # pst      = 01
0107a: 2f00          mov.l  %d0,-(%sp)    # pst      = 01
0107c: 302f 0008      mov.w  (8,%sp),%d0    # pst      = 01
01080: e488          lsr.l  &2,%d0       # pst      = 01
01082: 0280 0000 00ff  andi.l &0xff,%d0     # pst      = 01
01088: 207c 0080 1400  mov.l  &int_count,%a0   # pst      = 01
0108e: 52b0 0c00      addq.l &1,(0,%a0,%d0.1*4) # pst      = 01
01092: 11c0 a021      mov.b  %d0,IGCR0+1.w  # pst      = 01, 08
                                                # ddata   = 30, 30
                                                #           wdata.b = 0x00
01096: 1038 a020      mov.b  IGCRO.w,%d0    # pst      = 01, 08
                                                # ddata   = 28, 21
                                                #           rdata.b = 0x18

0109a: 4e71          nop                # pst      = 01
0109c: 71b8 ffe0      mvz.b  SWIACK.w,%d0    # pst      = 01, 08
                                                # ddata   = 20, 20
                                                #           rdata.b = 0x00

010a0: 0c80 0000 0041  cmpi.l %d0,&0x41     # pst      = 01
010a6: 6f08          ble.b  _isr_exit     # pst      = 05 (taken branch)
010b0: 201f          mov.l  (%sp)+,%d0    # pst      = 01
010b2: 205f          mov.l  (%sp)+,%a0    # pst      = 01

```

```
010b4: 4e73          rte
                      # pst    = 07, 03, 05, 0d
                      # ddata = 29, 21, 2a, 22
                      #
                      #      trg_addr = 2a19 << 1
                      #      trg_addr = 5432
```

As the PSTs are compressed, the resulting stream of 6-bit hexadecimal entries is loaded into consecutive locations in the PST trace buffer:

```
PSTB[*]= 1c, 1c, 05, 0d, // interrupt exception
         2a, 23, 28, 20, // branch target addr = 1074
         1a,             // 10 sequential insts
         13,             // 3 sequential insts
         05, 12,         // taken_branch + 2 sequential
         07, 03, 05, 0d, // rte, entry into user mode
         29, 21, 2a, 22 // branch target addr = 5432
```

Architectural studies on the compression algorithm determined an appropriate size for the PST trace buffer. Using a suite of ten MCU benchmarks, a 64-entry PSTB was found to capture an average window of time of 520 processor cycles with program trace using 2-byte addresses enabled.

#### 20.4.3.4 Processor Status, Debug Data 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:

PST = 0x01, {PST = 0x0[89B], DDATA = operand}

where the {...} definition is optional operand information defined by the setting of the CSR, and [...] indicates the presence of one value from the list.

The CSR provides capabilities to display operands based on reference type (read, write, or both). A PST value {0x08, 0x09, or 0x0B} identifies the size and presence of valid data to follow in the PST trace buffer (PSTB) {1, 2, or 4 bytes, respectively}. Additionally, CSR[DDC] specifies whether operand data capture is enabled and what size. Also, for certain change-of-flow instructions, CSR[BTB] provides the capability to display the target instruction address in the PSTB (2 or 3 bytes) using a PST value of 0x0D or 0x0E, respectively.

##### 20.4.3.4.1 User Instruction Set

[Table 20-27](#) 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 20-27. PST/DDATA Specification for User-Mode Instructions**

Instruction	Operand Syntax	PST/DDATA
add.l	<ea>y,Dx	PST = 0x01, {PST = 0x0B, DD = source operand}
add.l	Dy,<ea>x	PST = 0x01, {PST = 0x0B, DD = source}, {PST = 0x0B, DD = destination}
adda.l	<ea>y,Ax	PST = 0x01, {PST = 0x0B, DD = source operand}

**Table 20-27. PST/DDATA Specification for User-Mode Instructions (continued)**

<b>Instruction</b>	<b>Operand Syntax</b>	<b>PST/DDATA</b>
addi.l	#<data>,Dx	PST = 0x01
addq.l	#<data>,<ea>x	PST = 0x01, {PST = 0x0B, DD = source}, {PST = 0x0B, DD = destination}
addr.l	Dy,Dx	PST = 0x01
and.l	<ea>y,Dx	PST = 0x01, {PST = 0x0B, DD = source operand}
and.l	Dy,<ea>x	PST = 0x01, {PST = 0x0B, DD = source}, {PST = 0x0B, DD = destination}
andi.l	#<data>,Dx	PST = 0x01
asl.l	{Dy,#<data>},Dx	PST = 0x01
asr.l	{Dy,#<data>},Dx	PST = 0x01
bcc.{b,w,l}		if taken, then PST = 0x05, else PST = 0x01
bchg.{b,l}	#<data>,<ea>x	PST = 0x01, {PST = 0x08, DD = source}, {PST = 0x08, DD = destination}
bchg.{b,l}	Dy,<ea>x	PST = 0x01, {PST = 0x08, DD = source}, {PST = 0x08, DD = destination}
bclr.{b,l}	#<data>,<ea>x	PST = 0x01, {PST = 0x08, DD = source}, {PST = 0x08, DD = destination}
bclr.{b,l}	Dy,<ea>x	PST = 0x01, {PST = 0x08, DD = source}, {PST = 0x08, DD = destination}
bitrev.l	Dx	PST = 0x01
bra.{b,w,l}		PST = 0x05
bset.{b,l}	#<data>,<ea>x	PST = 0x01, {PST = 0x08, DD = source}, {PST = 0x08, DD = destination}
bset.{b,l}	Dy,<ea>x	PST = 0x01, {PST = 0x08, DD = source}, {PST = 0x08, DD = destination}
bsr.{b,w,l}		PST = 0x05, {PST = 0x0B, DD = destination operand}
btst.{b,l}	#<data>,<ea>x	PST = 0x01, {PST = 0x08, DD = source operand}
btst.{b,l}	Dy,<ea>x	PST = 0x01, {PST = 0x08, DD = source operand}
byterev.l	Dx	PST = 0x01
clr.b	<ea>x	PST = 0x01, {PST = 0x08, DD = destination operand}
clr.l	<ea>x	PST = 0x01, {PST = 0x0B, DD = destination operand}
clr.w	<ea>x	PST = 0x01, {PST = 0x09, DD = destination operand}
cmp.b	<ea>y,Dx	PST = 0x01, {0x08, source operand}
cmp.l	<ea>y,Dx	PST = 0x01, {PST = 0x0B, DD = source operand}
cmp.w	<ea>y,Dx	PST = 0x01, {0x09, source operand}
cmpa.l	<ea>y,Ax	PST = 0x01, {PST = 0x0B, DD = source operand}
cmpa.w	<ea>y,Ax	PST = 0x01, {0x09, source operand}
cmpl.b	#<data>,Dx	PST = 0x01
cmpl.l	#<data>,Dx	PST = 0x01
cmpl.w	#<data>,Dx	PST = 0x01
eor.l	Dy,<ea>x	PST = 0x01, {PST = 0x0B, DD = source}, {PST = 0x0B, DD = destination}

**Table 20-27. PST/DDATA Specification for User-Mode Instructions (continued)**

<b>Instruction</b>	<b>Operand Syntax</b>	<b>PST/DDATA</b>
eori.l	#<data>,Dx	PST = 0x01
ext.l	Dx	PST = 0x01
ext.w	Dx	PST = 0x01
extb.l	Dx	PST = 0x01
illegal		PST = 0x01 <sup>1</sup>
jmp	<ea>y	PST = 0x05, {PST = 0x0[DE], DD = target address} <sup>2</sup>
jsr	<ea>y	PST = 0x05, {PST = 0x0[DE], DD = target address}, {PST = 0x0B, DD = destination operand} <sup>2</sup>
lea.l	<ea>y,Ax	PST = 0x01
link.w	Ay,#<displacement>	PST = 0x01, {PST = 0x0B, DD = destination operand}
lsl.l	{Dy,#<data>},Dx	PST = 0x01
lsr.l	{Dy,#<data>},Dx	PST = 0x01
mov3q.l	#<data>,<ea>x	PST = 0x01, {PST = 0x0B, DD = destination operand}
move.b	<ea>y,<ea>x	PST = 0x01, {PST = 0x08, DD = source}, {PST = 0x08, DD = destination}
move.l	<ea>y,<ea>x	PST = 0x01, {PST = 0x0B, DD = source}, {PST = 0x0B, DD = destination}
move.w	<ea>y,<ea>x	PST = 0x01, {PST = 0x09, DD = source}, {PST = 0x09, DD = destination}
move.w	CCR,Dx	PST = 0x01
move.w	{Dy,#<data>},CCR	PST = 0x01
movea.l	<ea>y,Ax	PST = 0x01, {PST = 0x0B, DD = source}
movea.w	<ea>y,Ax	PST = 0x01, {PST = 0x09, DD = source}
movem.l	#list,<ea>x	PST = 0x01, {PST = 0x0B, DD = destination},...
movem.l	<ea>y,#list	PST = 0x01, {PST = 0x0B, DD = source},...
moveq.l	#<data>,Dx	PST = 0x01
muls.l	<ea>y,Dx	PST = 0x01, {PST = 0x0B, DD = source operand}
muls.w	<ea>y,Dx	PST = 0x01, {PST = 0x09, DD = source operand}
mulu.l	<ea>y,Dx	PST = 0x01, {PST = 0x0B, DD = source operand}
mulu.w	<ea>y,Dx	PST = 0x01, {PST = 0x09, DD = source operand}
mvs.b	<ea>y,Dx	PST = 0x01, {0x08, source operand}
mvs.w	<ea>y,Dx	PST = 0x01, {0x09, source operand}
mvz.b	<ea>y,Dx	PST = 0x01, {0x08, source operand}
mvz.w	<ea>y,Dx	PST = 0x01, {0x09, source operand}
neg.l	Dx	PST = 0x01
negx.l	Dx	PST = 0x01

**Table 20-27. PST/DDATA Specification for User-Mode Instructions (continued)**

<b>Instruction</b>	<b>Operand Syntax</b>	<b>PST/DDATA</b>
nop		PST = 0x01
not.l	Dx	PST = 0x01
or.l	<ea>y,Dx	PST = 0x01, {PST = 0x0B, DD = source operand}
or.l	Dy,<ea>x	PST = 0x01, {PST = 0x0B, DD = source}, {PST = 0x0B, DD = destination}
ori.l	#<data>,Dx	PST = 0x01
pea.l	<ea>y	PST = 0x01, {PST = 0x0B, DD = destination operand}
pulse		PST = 0x04
rts		PST = 0x01, {PST = 0x0B, DD = source operand}, PST = 0x05, {PST = 0x0[DE], DD = target address}
sats.l	Dx	PST = 0x01
scc.b	Dx	PST = 0x01
sub.l	<ea>y,Dx	PST = 0x01, {PST = 0x0B, DD = source operand}
sub.l	Dy,<ea>x	PST = 0x01, {PST = 0x0B, DD = source}, {PST = 0x0B, DD = destination}
suba.l	<ea>y,Ax	PST = 0x01, {PST = 0x0B, DD = source operand}
subi.l	#<data>,Dx	PST = 0x01
subq.l	#<data>,<ea>x	PST = 0x01, {PST = 0x0B, DD = source}, {PST = 0x0B, DD = destination}
subx.l	Dy,Dx	PST = 0x01
swap.w	Dx	PST = 0x01
tas.b	<ea>x	PST = 0x01, {0x08, source}, {0x08, destination}
tpf		PST = 0x01
tpf.l	#<data>	PST = 0x01
tpf.w	#<data>	PST = 0x01
trap	#<data>	PST = 0x01 <sup>1</sup>
tst.b	<ea>x	PST = 0x01, {PST = 0x08, DD = source operand}
tst.l	<ea>y	PST = 0x01, {PST = 0x0B, DD = source operand}
tst.w	<ea>y	PST = 0x01, {PST = 0x09, DD = source operand}
unlk	Ax	PST = 0x01, {PST = 0x0B, DD = destination operand}
wddata.b	<ea>y	PST = 0x04, {PST = 0x08, DD = source operand}
wddata.l	<ea>y	PST = 0x04, {PST = 0x0B, DD = source operand}
wddata.w	<ea>y	PST = 0x04, {PST = 0x09, DD = source operand}

- <sup>1</sup> During normal exception processing, the PSTB is loaded with two successive 0x1C entries 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 Processing:

```
PST = 0x1C, 0x1C,  
{PST = 0x0B, DD = destination}, // stack frame  
{PST = 0x0B, DD = destination}, // stack frame  
{PST = 0x0B, DD = source}, // vector read  
PST = 0x05, {PST = 0x0[DE], DD = target} // handler PC
```

A similar set of PST/DD values is generated in response to an emulator mode exception. For these events (caused by a debug interrupt or properly-enabled trace exception), the initial PST values are 0x1D, 0x1D and the remaining sequence is equivalent to normal exception processing.

The PST/DDATA specification for the reset exception is shown below:

Exception Processing:

```
PST = 0x1C, 0x1C,  
PST = 0x05, {PST = 0x0[DE], DD = target} // initial PC
```

The initial references at address 0 and 4 are never captured nor displayed because these accesses are treated as instruction fetches.

For all types of exception processing, the PST = 0x1C (or 0x1D) value is driven for two trace buffer entries.

- <sup>2</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).

#### 20.4.3.4.2 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 20-28](#).

**Table 20-28. PST/DDATA Specification for Supervisor-Mode Instructions**

Instruction	Operand Syntax	PST/DDATA
halt		PST = 0x1F, PST = 0x1F
move.l	Ay,USP	PST = 0x01
move.l	USP,Ax	PST = 0x01
move.w	SR,Dx	PST = 0x01
move.w	{Dy,#<data>},SR	PST = 0x01, {PST = 0x03}
movec.l	Ry,Rc	PST = 0x01
rte		PST = 0x07, {PST = 0x0B, DD = source operand}, {PST = 0x03}, {PST = 0x0B, DD = source operand}, PST = 0x05, {PST = 0x0[DE], DD = target address}
stldsr.w	#imm	PST = 0x01, {PST = 0x0B, DD = destination operand, PST = 0x03}
stop	#<data>	PST = 0x1E, PST = 0x1E
wdebug.l	<ea>y	PST = 0x01, {PST = 0x0B, DD = source, PST = 0x0B, DD = source}

The move-to-SR, STLDSR, and RTE instructions include an optional PST = 0x3 value, indicating an entry into user mode.

Similar to the exception processing mode, the stopped state (PST = 0x1E) and the halted state (PST = 0x1F) display this status for two entries when the ColdFire processor enters the given mode.

#### 20.4.4 Freescale-Recommended BDM Pinout

Typically, a relatively simple interface pod is used to translate commands from a host computer into commands for the custom serial interface to the single-wire background debug system. Depending on the development tool vendor, this interface pod may use a standard RS-232 serial port, a parallel printer port, or some other type of communications such as a universal serial bus (USB) to communicate between the host PC and the pod. The pod typically connects to the target system with ground, the BKGD pin,  $\overline{\text{RESET}}$ , and sometimes  $V_{DD}$ . An open-drain connection to reset allows the host to force a target system reset, useful to regain control of a lost target system or to control startup of a target system before the on-chip nonvolatile memory has been programmed. Sometimes  $V_{DD}$  can be used to allow the pod to use power from the target system to avoid the need for a separate power supply. However, if the pod is powered separately, it can be connected to a running target system without forcing a target system reset or otherwise disturbing the running application program.

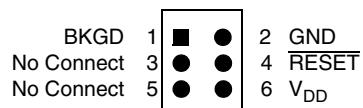


Figure 20-26. Recommended BDM Connector

# Appendix A

## Revision History

This appendix lists major changes between versions of the MCF51CN128RM document.

### A.1 Changes Between Rev. 2 and Rev. 3

Table 20-29. MCF51CN128RM Rev. 2 to Rev. 3 Changes

Chapter	Description
Parallel/Input/Output Controls	Removed set/clear/toggle functionality for GPIO only. This functionality is available for GPIO.
Parallel Input/Output, Rapid GPIO (GPIO), Analog-to-Digital Converter (ADC12), Real Time Counter (RTC), Serial Communication Interface (SCI), Serial Peripheral Interface (SPI), Inter-Integrated Circuit (IIC), Mini-FlexBus, Fast Ethernet Controller (FEC), Timer/PWM Module (TPM), and Modulo Timer (MTIM)	In chapter introduction sections, added GPIO note, 'Use pin mux control registers from <a href="#">Section 2.3, "Pin Mux Controls"</a> to assign GPIO signals to the MCF51CN128 package pins. Most pin functions default to GPIO and must be software configured.'
Memory, Interrupt Controller (CF1_INTC)	Updated the representation of INTC_ORMR register.
Version 1 ColdFire Debug (CF1_DEBUG)	In introduction section, added note, 'Use pin mux control registers from <a href="#">Section 2.3, "Pin Mux Controls"</a> to assign the debug signals to the device's package pins.'

### A.2 Changes Between Rev. 3 and Rev. 4

Table 20-30. MCF51CN128RM Rev. 3 to Rev. 4 Changes

Chapter	Description
Book Title	Removed MCF51CN64 from devices supported.
About This Book	Removed MCF51CN64 instances.
Device Overview	Removed MCF51CN64 from <a href="#">Table 1-2</a> .
Memory	Removed MCF51CN64 device instances from <a href="#">Section 4.4.1, "Features."</a>

### A.3 Changes Between Rev. 4 and Rev. 5

**Table 20-31. MCF51CN128RM Rev. 4 to Rev. 5 Changes**

Chapter	Description
Memory	In <a href="#">Table 4-3</a> , changed bit 7 of the SOPT3 register to 0 and grayed out.
Analog-to-Digital Converter (ADC12)	In <a href="#">Section 15.4.4.5, “Sample Time and Total Conversion Time,”</a> changed the unit of conversion time to $\mu\text{s}$ .
Modulo Timer (MTIM)	Added a note in introduction section to ignore references to stop1.
Serial Communication Interface (SCI)	Changed bus clock to SCI module clock.
Timer/PWM Module (TPM)	Updated the description of fixed frequency clock.

### A.4 Changes Between Rev. 5 and Rev. 6

**Table 20-32. MCF51CN128RM Rev. 5 to Rev. 6 Changes**

Chapter	Description
Modes of Operation	In <a href="#">Table 3-5</a> , updated RTC operation in Stop4 mode.
Memory	In <a href="#">Table 4-3</a> , updated Mini-flexbus register bits and added section <a href="#">4.2.1/4-22</a> .
Version 1 ColdFire Debug (CF1_DEBUG)	Updated Note in <a href="#">20.3.8/20-22</a> to, “Version 1 ColdFire core devices implement a 24-bit, 16 MB address map. When programming these registers with a 32-bit address, the upper byte must be zero-filled.”